DECgraphic-11

FORTRAN
Reference Manual

Order No. DEC-11-GFRMA-A-D

digital equipment corporation · maynard. massachusetts

First Printing, November 1976

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1976 by Digital Equipment Corporation

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECsystem-20	TYPESET-11

CONTENTS

			Page
PREFACE			vii
CHAPTER	1	THE DECGRAPHIC-11 SYSTEMS	1-1
	1.1	AN INTRODUCTION TO DECGRAPHIC-11	1-1
	1.1.1	Overview of the Manual	1-1
	1.1.2	Documentation Conventions	1-2
	1.1.3	Overview of DECgraphic-11 System Capabilities	1-3
	1.2	HARDWARE/SOFTWARE ENVIRONMENT	1-6
	1.3	BASIC GRAPHICS CONCEPTS	1-7
	1.3.1	The Display Screen	1-7
	1.3.2	The Display File	1-9
	1.3.3	Display Primitives	1-11
	1.3.4	Subpicture Definitions	1-12
	1.3.5	Display File Pointers	1-13
	1.3.6	User Coordinate Systems and Windows	1-14
	1.3.7	Light Pen Interaction and Tracking	1-15
	1.4	GRAPHICS SUBROUTINES	1-18
	1.4.1	Initializing and Controlling the Display File	1-19
	1.4.2	Setting Screen and Scaling Parameters	1-19
	1.4.3	Generating Graphics Primitives	1-20
	1.4.4	Defining and Using Subpictures	1-21
	1.4.5	Displaying Graphs and Figures	1-23
	1.4.6	Using Display File Pointers	1-23
	1.4.7	Altering Display File Status Parameters	1-25
	1.4.8	Facilitating Light Pen Interaction	1-25
	1.4.9	Performing Display File Utility Functions	1-27
	1.4.10	Performing Advanced Display File Functions	1-27
CHAPTER	2	DECGRAPHIC-11 GRAPHICS SUBROUTINES	2-1
	2.1	INITIALIZING AND CONTROLLING THE DISPLAY FILE	2-1
	2.1.1	INIT: Initializing the Display File	2-1
	2.1.2	STOP: Stopping the Display	2-3
	2.1.3	CONT: Restoring the Display	2-3
	2.1.4	FREE: Clearing the Display File Area	2-4
	2.2	SETTING SCREEN AND SCALING PARAMETERS	2-4
	2.2.1	SCOPE: Selecting a Display Scope	2-4
	2.2.2	AREA: Selecting the Main or Menu Area	2-4
	2.2.3	SCAL: Defining a User Coordinate System	2-5
	2.2.4	NOSC: Restoring the Standard Coordinate System	2-7
	2.2.5	WINDOW: Establishing a Display Screen Window	2-7
	2.3	GENERATING GRAPHICS PRIMITIVES	2-9
	2.3.1	APNT: Displaying an Absolute Point	2-10
	2.3.2	RPNT: Displaying a Relative Point	2-10

		Page
2.3.3	VECT: Drawing a Relative Vector	2-11
2.3.4	AVECT: Drawing an Absolute Vector	2-12
2.3.5	SVECT: Drawing a Vector in Short Format	2-12
2.3.6	LVECT: Drawing a Vector in Long Format	2-13
2.3.7	TEXT: Displaying a Character String	2-13
2.3.8	MENU: Displaying Items in the Menu Area	2-18
2.4	DEFINING AND USING SUBPICTURES	2-19
2.4.1	SUBP: Defining a Subpicture	2-19
2.4.2	ESUB: Terminating a Subpicture	2-21
2.4.3	COPY: Copying a Subpicture	2-21
2.4.4	OFF: Turning Off a Subpicture	2-22
2.4.5	ON: Turning on a Subpicture	2-23
2.4.6	ERAS: Erasing a Subpicture	2-23
2.4.7	NMBR: Creating a Numeric Subpicture	2-24
2.4.8	CVSCAL: Scaling Subpicture Characters and Vectors	2-25
2.5	DISPLAYING GRAPHS AND FIGURES	2-27
2.5.1	XGRA: Displaying an X-Axis Graph	2-27
2.5.2	YGRA: Displaying a Y-Axis Graph	2-27
2.5.3	FIGR: Displaying a Figure	2-28
2.5.4	AGET: Returning the Value of a Primitive	2-29
2.5.5	APUT: Changing the Value of a Primitive	2-29
2.5.6	FPUT: Changing and Adjusting the Value of a Primitive	2-30
2.6	USING DISPLAY FILE POINTERS	2-30
2.6.1	POINTR: Setting Up a Pointer	2-30
2.6.2	ADVANC: Advancing a Pointer	2-31
2.6.3	GET: Returning the Coordinates of a Primitive	2-31
2.6.4	CHANGE: Changing the Coordinates of a Primitive	2-32
2.6.5	CHANGA: Changing a Primitive and Adjusting the Next Primitive	2-32
2.6.6	CHANGT: Changing the Value of a Character Primitive	2-33
2.6.7	INSERT: Inserting Graphic Elements in the Display File	2-34
2.6.8	ERASP: Erasing a Primitive	2-34
2.7	ALTERING DISPLAY FILE STATUS PARAMETERS	2-35
2.7.1	SENSE: Setting the Light Pen Parameter	2-35
2.7.2	INTENS: Setting the Intensity Parameter	2-36
2.7.3	FLASH: Setting the Flash-Mode Parameter	2-36
2.7.4	LINTYP: Setting the Line-Type Parameter	2-37
2.8	FACILITATING LIGHT PEN INTERACTION	2-37
2.8.1	LPEN: Recording a Light Pen Hit	2-37
2.8.2	TRAK: Placing a Tracking Object on the Screen	2-38
2.8.3	TRAKXY: Returning the Position of the Tracking Object	2-39
2.8.4	ATTACH: Attaching a Primitive to the Tracking Object	2-39
2.8.5	DETACH: Detaching Primitives from the Tracking Object	2-40
2.8.6	GRID: Positioning the Tracking Object on the Grid	2-40
2.9	PERFORMING DISPLAY FILE UTILITY FUNCTIONS	2-41
2.9.1	CMPRS: Compressing the Display File	2-41
2.9.2	SAVE: Saving the Display File	2-42

			Page
	2.9.3	RSTR: Restoring the Display File	2-43
	2.10	PERFORMING ADVANCED DISPLAY FILE FUNCTIONS	2-44
	2.10.1	DPTR: Returning the Next Available Display File Position	2-45
	2.10.2	DPYNOP: Inserting No-ops in the Display File	2-45
	2.10.3	DPYWD: Inserting a Data Word in the Display File	2-45
CHAPTER	3	PROGRAMMING TECHNIQUES	3-1
	3.1	SUBPICTURE TECHNIQUES	3-1
	3.1.1	Using Subpictures as Subroutines	3-1
	3.1.2	Creating Complete Displays	3-2
	3.1.3	Attaching a Subpicture	3-2
	3.1.4	Using NMBR for Odometer Output	3-2
	3.2	GENERAL GRAPHICS TECHNIQUES	3-3
	3.2.1	Specifying Vector Formats	3-3
	3.2.2	Ordering Display Elements	3-4
	3.2.3	Controlling Display File Size	3-4
CHAPTER	4	THE RT-11 OPERATING ENVIRONMENT	4-1
	4.1	BUILDING THE DECGRAPHIC-11 LIBRARIES	4-1
	4.1.1	Binary Kit	4-1
	4.1.2	Source Kit	4-2
	4.2	LINKING USER PROGRAMS	4-3
	4.3	PERFORMING USR OPERATIONS	4-4
	4.4	SAMPLE PROCEDURES	4-5
	4.4.1	VT11 Procedures	4-5
	4.4.2	VS60 Procedures	4-6
CHAPTER	5	THE RSX-11M OPERATING ENVIRONMENT	5-1
	5.1	INTRODUCTION TO SUPPLIED SOFTWARE	5-1
	5.2	OPERATION UNDER RSX-11M	5-1
	5.2.1	Building Your DECgraphic-11 Library	5-2
	5.2.2	Writing and Running Your Own DECgraphic-11 Programs	5-3
APPENDIX	A	DECGRAPHIC-11 SUBROUTINE SUMMARY	A-1
APPENDIX	B	DECGRAPHIC-11 ERROR MESSAGES	B-1
APPENDIX	C	DISPLAY FILE STRUCTURE	C-1
APPENDIX	D	FORTRAN PROGRAMMING EXAMPLE	D-1
APPENDIX	E	DIFFERENCES BETWEEN THE DECGRAPHIC-11 AND RT-11 GRAPHICS EXTENSIONS PACKAGES	E-1
APPENDIX	F	DIFFERENCES BETWEEN DECGRAPHIC-11 AND RSX-11M/FORTRAN GRAPHIC EXTENSIONS	F-1
GLOSSARY			
INDEX			

Page

FIGURES

FIGURE	1-1	Line Types	1-3
	1-2	Intensity Levels	1-4
	1-3	Type Fonts	1-4
	1-4	Extended Characters	1-5
	1-5	Rotated Characters	1-5
	1-6	Subscripts and Superscripts	1-5
	1-7	Character Scaling	1-6
	1-8	VS60 Display Screen	1-8
	1-9	Menu Area	1-9
	1-10	CPU and DPU	1-10
	1-11	Viewing Window	1-15
	1-12	Light Buttons	1-16
	1-13	Tracking Object	1-17
	2-1	WINDOW Subroutine	2-8
	2-2	MENU Subroutine	2-18
	2-3	YGRA Subroutine	2-28

PREFACE

DECgraphic-11 is the family name for PDP-11 graphic products using the VT11 and VS60 graphic display systems. Each system consists of a display processor on the UNIBUS which controls a cathode ray tube display with light pen. Installations of this type have a wide range of applications, including simulation studies, computer-aided design, and real-time data acquisition. The UNIBUS architecture permits a DECgraphic-11 system to function as a small terminal or as part of a larger stand-alone disk-based system.

DECgraphic-11 software is made up of sets of FORTRAN-callable subroutines which give the user full access to the system's powerful graphic capabilities. The use of the routines is in keeping with the usual procedures in FORTRAN programming, which makes the power of DECgraphic-11 applicable to the widest range of foreseeable applications. By a simple procedure, the user can initialize the software to comprise only those routines needed for a particular application and hardware configuration.

This manual is intended for DECgraphic-11 users who are familiar with PDP-11 FORTRAN under the RT-11 or RSX-11M operating systems. It introduces the characteristics of the VT11 and VS60 display processors, describes all of the FORTRAN-callable subroutines, and summarizes the operation of DECgraphic-11 in the environment of each operating system. It does not provide comprehensive information on the FORTRAN language or on the system resources of RT-11 or RSX-11M. Information on these and related topics can be found in the manuals listed below.

ASSOCIATED DOCUMENTATION

PDP-11 FORTRAN Language Reference Manual
DEDC-11-LFLRA-C-D

RT-11/RSTS/E FORTRAN IV User's Guide
DEC-11-LRRUA-A-D

RT-11 System Reference Manual
DEC-11-ORUGA-C-D, DN1, DN2

IAS/RSX-11 FORTRAN IV User's Guide
DEC-11-LMFUA-C-D

RSX-11M Operator's Procedures Manual
DEC-11-OMOGA-B-D

RSX-11M Utilities Procedures Manual
DEC-11-OMUPA-B-D

RSX-11M Executive Reference Manual
DEC-11-OMERA-B-D

RSX-11M Task Builder Reference Manual
DEC-11-OMTBA-B-D

VT11 Graphic Display Monitor User's Manual
EK-VT11-TM-001

GT40/GT42 User's Guide
EK-GT40-OP-002

The routines in this manual are largely compatible with, but more extensive than, earlier sets of FORTRAN-callable routines described in the FORTRAN/RT-11 Extensions Manaul (DEC-11-LRTEA-C-D) and the RSX-11M Graphics Extensions User's Guide (DEC-11-AMLEA-A-D).

CHAPTER 1

THE DECGRAPHIC-11 SYSTEMS

1.1 AN INTRODUCTION TO DECGRAPHIC-11

DECgraphic-11 graphics systems provide comprehensive hardware and software interactive graphics facilities for PDP-11 users. The DECgraphic-11 FORTRAN support package allows programmers to use powerful graphics software capabilities for the VT11 and VS60 display subsystems. This graphics package consists of a collection of subroutines that may be issued from FORTRAN programs to perform a variety of graphics operations. The subsections that follow provide an overview of this manual, describe the documentation conventions used in the presentation of subroutines, and supply an introduction to the varied capabilities offered by the DECgraphic-11 systems.

1.1.1 Overview Of The Manual

This manual describes the use of DECgraphic-11 FORTRAN support in two operating environments. It is divided into five chapters.

Chapter 1 introduces the basic graphics concepts necessary for an understanding of DECgraphic-11 system operations. It defines the terms used in the manual, discusses the characteristics of the VT11 and VS60 display processors, and summarizes the use of the FORTRAN-callable subroutines in a sequence of 10 functional categories.

Chapter 2 describes the DECgraphic-11 FORTRAN-callable subroutines in detail and provides a variety of examples of their use. Most of the subroutines are used interchangeably for the VT11 and VS60 displays. Where differences exist--for example, in cases where routines invoke advanced VS60 hardware features-- these differences are clearly distinguished.

Chapter 3 provides the graphics user with a variety of programming techniques to aid in the efficient use of the DECgraphic-11 system capabilities.

Chapter 4 describes the operation of DECgraphic-11 FORTRAN support in an RT-11 environment. It includes information on building a library of FORTRAN subroutines, describes a method for linking user programs, and provides sample VT11 and VS60 build procedures.

Chapter 5 discusses DECgraphic-11 operation in the RSX-11M environment, including procedures for building a DECgraphic-11 library, compiling the user's programs, and building them into task images.

In addition, a series of appendices provides summary information as a reference aid to DECgraphic-11 users.

Appendix A provides an alphabetical list of FORTRAN subroutine calls and briefly describes the function of each.

Appendix B summarizes the error messages that might be generated by the DECgraphic-11 package, the reasons for occurrence, and routines that might produce these errors.

Appendix C discusses the internal format of the DECgraphic-11 display file as it is constructed in memory.

Appendix D is a FORTRAN programming example, developed to demonstrate the varied capabilities of the DECgraphic-11 interactive graphics system.

Appendix E summarizes the differences between the DECgraphic-11 subroutines and the subroutines supplied in the RT-11/FORTRAN Graphics Extensions package.

Appendix F describes the differences between DECgraphic-11 subroutines and the subroutines in the RSX-11M/FORTRAN graphic extensions.

Finally, a glossary of graphics terms defines the various special terms and concepts used in this manual.

1.1.2 Documentation Conventions

The following list describes the documentation conventions used in describing the FORTRAN graphics calls in this manual.

Convention	Meaning
Square brackets ([ ])	The enclosed parameters are optional.
FORTRAN variable names with upper-case letters (X11, I0, M2)	Standard default FORTRAN variable types whose value is returned by the called routine; alternatively, an array name passed to the subroutine.
FORTRAN variable names with lower-case letters (x11, i0, m2)	Standard default FORTRAN variable types whose value is to be supplied by the user; may be any valid arithmetic expression of the specified type (i.e., x is real, i is an integer).
[l[,i[,f[,t]]]]	The only exceptions to the rules described above; special display file status parameters that are all integers (see Section 2.3); in subroutine calls, expressed only as [l, i, f, t].
x-axis	The horizontal axis.
y-axis	The vertical axis.

NOTE

All floating-point constants supplied by the user as arguments in calls to the FORTRAN subroutines should have decimal points. If you do not adhere strictly to this convention, the results of program execution are unpredictable.

1.1.3 Overview of DECgraphic-11 System Capabilities

The following list summarizes the major user-visible capabilities of the DECgraphic-11 interactive graphics system. Graphics terms such as beam, vector, menu, and so on are defined in a glossary at the end of this manual and are explained in some detail in Section 1.3.

Support of varied display elements

You may display a variety of different types of elements on the screen, including points, line segments, characters, and graphs. Elements are normally defined at coordinate positions relative to the current beam position, although points can be defined at absolute coordinate positions, as can line segments on the VS60.

User-selected line types

Vectors may be drawn on the screen in any of four formats: solid line, long-dashed line, short-dashed line, or dot-dash line. These are illustrated in Figure 1-1.

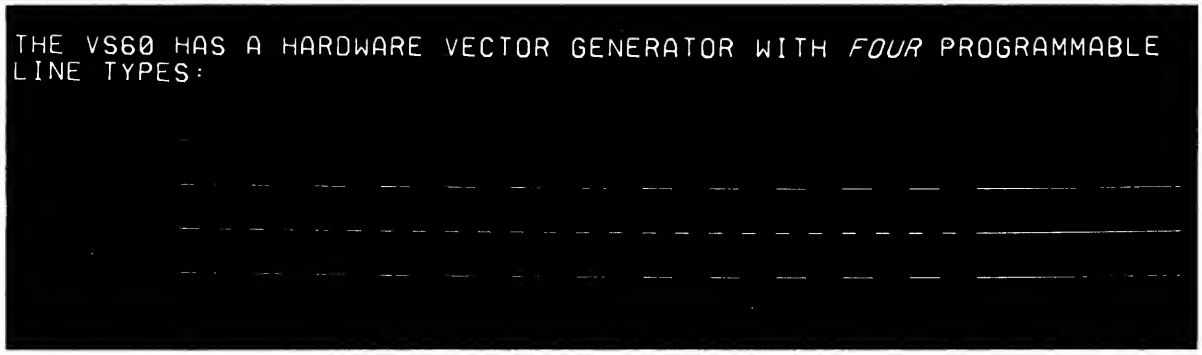

Figure 1-1 Line Types

User-selected intensity level

You may vary the brightness of pictures or components of pictures on the display screen. In this way, you can emphasize portions of the display. There are eight different levels of brightness, from 1 (faintest) to 8 (brightest), as shown in Figure 1-2.

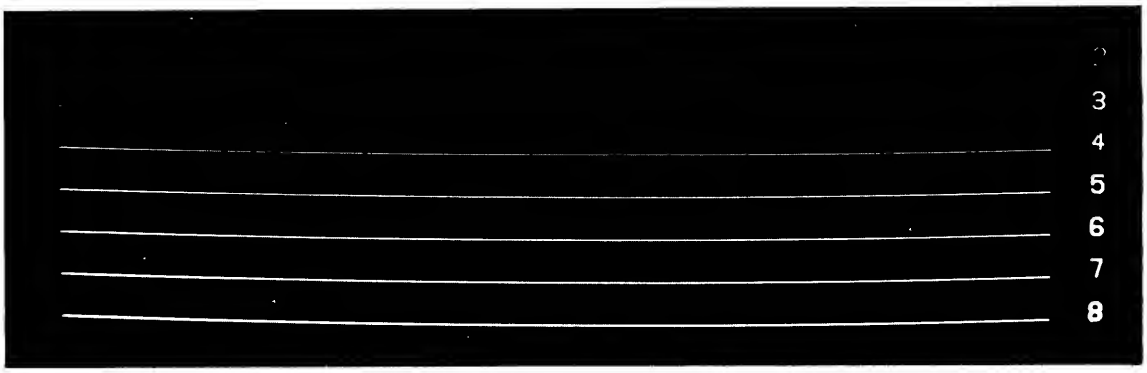

Figure 1-2 Intensity Levels

Hardware blink feature

You may specify that a display or a portion of a display is to blink on and off. This is useful in providing user warnings or in identifying a particular picture component to be changed or highlighted. This feature is also known as flash mode.

Italic mode

Characters may be displayed in ordinary type or in italics. You may specify the text to be italicized by including a special control code before the character string specification. An example of the two available type fonts is shown in Figure 1-3.

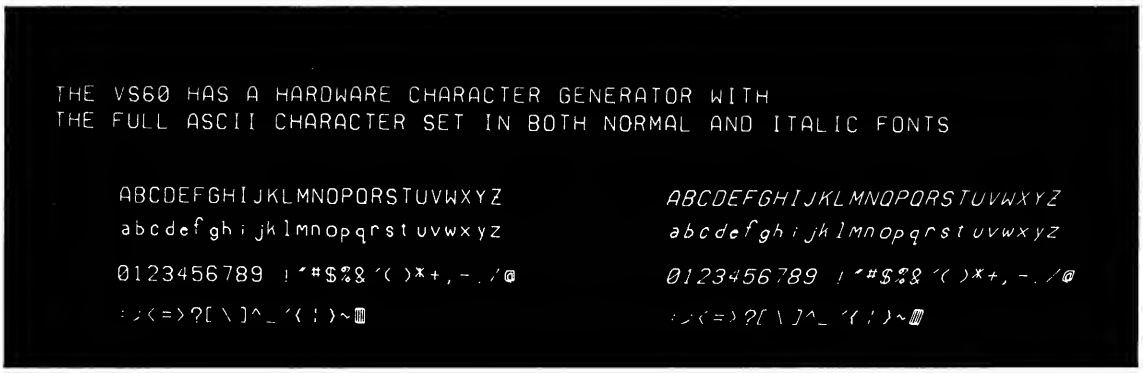

Figure 1-3 Type Fonts

Extended character set

You may output any of the conventional 96 ASCII characters and may also display 31 additional characters representing Greek letters and mathematical symbols. To select characters from this extended set, you must include a "shift-out" code before the ordinary ASCII character string to which the shift-out character string corresponds. Extended characters may be displayed in either of the available type fonts, as shown in Figure 1-4.

Figure 1-4 Extended Characters

A list of extended characters and their ASCII character correspondences is included in Section 2.3.7.

Rotated characters

On the VS60, a character string may be displayed in the normal horizontal position or may be rotated 90 degrees counter-clockwise. As in the specification of italic mode, you may select rotated characters by including a special control code before the character string to be rotated. Examples of normal and rotated characters are shown in Figure 1-5.

Figure 1-5 Rotated Characters

Subscript and superscript generation

On the VS60, you may display text that incorporates subscripts and superscripts on the screen--for example, in the expression shown in Figure 1-6.

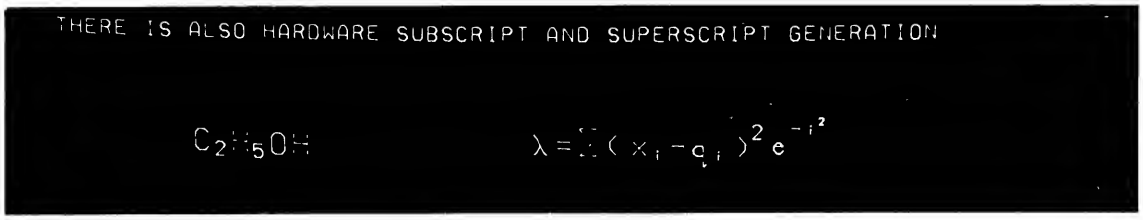

Figure 1-6 Subscripts and Superscripts

As for other special character modes, you specify a subscript or superscript by including a special control code before the character string to be displayed.

Character and image scaling

On the VS60, you may display characters and lines on the screen in the standard size or may enlarge or compress them as desired for display purposes. The normal size of each character is 6 x 8 raster units. You may change this standard character size and display characters that are one-half normal size or one and one-half or twice normal size, as shown in the example of Figure 1-7. Vectors and other images may be scaled in increments of one-fourth from one-fourth normal size to three and three-fourths normal size.

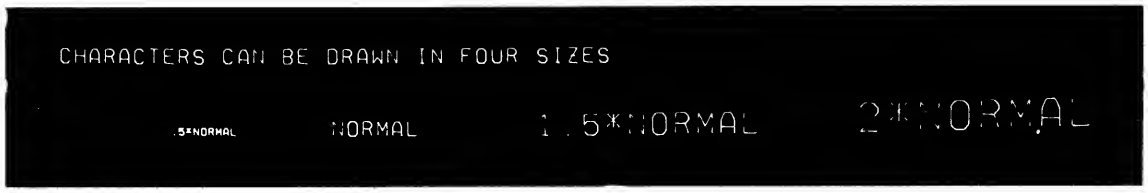

Figure 1-7 Character Scaling

Scaling and windowing

On the VS60, you may define the screen to function as a viewing window on a much larger image-definition or drawing area for the display image (see Section 1.3). You can vary the position of this "window" and in this way examine different areas of the total image. You may also define your own coordinate system.

Light pen support

You may interact with the display processor by using the light pen to select options, identify pictures to be moved, or otherwise manipulate displays on the screen. Support of the light pen on the VT11 and VS60 and the light pen tip switch on the VS60 facilitate the dynamic alteration of displays on the screen and make DECgraphic-11 a truly interactive system.

Menu area

You may display a special menu of character strings to be displayed in a formatted list on the screen and selected by pressing the light pen down at the location of the desired option. You may specify your own menu area or on the VS60 may use the hardware menu area, described in Section 1.3.1.

Subpicture support

Internally, repeated displays of identical images can be defined with more modularity and efficiency by making use of subpicture support. You can group a series of calls, data, and picture components together to form an entity called a subpicture, which can then be displayed, copied, erased, and turned on and off.

1.2 HARDWARE/SOFTWARE ENVIRONMENT

The hardware normally required for DECgraphic-11 FORTRAN support includes the following:

- PDP-11 central processing unit with 16K or more of memory
- VS60 or VT11 display subsystem
- user terminal
- disk or another random-access system storage device for use by the operating system

The amount of memory required to support the graphics system is dependent on your own programming requirements, but will normally range from 8K to 16K.

The VS60 and VT11 are supported as autonomous processors, attached as UNIBUS peripherals to the PDP-11. Both processors are direct-memory-access devices.

1.3 BASIC GRAPHICS CONCEPTS

This section describes the most important of the graphics concepts that underlie the operation of the DECgraphic-11 FORTRAN support package. It provides background information in the use of display screens and light pens, the construction and use of display files, the creation of display pictures from points and line segments, and the use of subpictures, pointers, and display parameters in the DECgraphic-11 system. Some of the information provided here expands upon the discussion of system capabilities in the introductory sections of this chapter. Understanding the concepts presented here is necessary to interpreting the operations of the subroutines summarized in Section 1.4 and described in detail in Chapter 2.

1.3.1 The Display Screen

Two graphics display subsystems may be used in DECgraphic-11 systems: the VS60 and the VT11. Both graphics subsystems have refresh CRT capabilities and support the use of a solid-state light pen. The VS60 display processor can support two distinct scopes.

The display screen provides a square main viewing area; this area is a 9 1/4-inch by 9 1/4-inch (20.74-centimeter by 20.74-centimeter) area on the VT11 and a 12-inch by 12-inch (30.48-centimeter by 30.48-centimeter) area on the VS60. It addresses 1024 by 1024 raster units. For the VS60, this viewing area may actually be considered a window on a larger image-definition area. The logical screen that can be addressed on the VS60 is a 96-inch by 96-inch (243.84-centimeter by 243.84-centimeter) area. Subsequent sections of this manual describe the ways in which you may manipulate the screen viewing area in order to examine a different portion of this larger image-definition area.

The main viewing area's lower left corner can be described by the coordinate positions (x=0, y=0); its upper right corner has coordinates (x=1023, y=1023). Both the x-axis and the y-axis consist of 1024 individual points (1777 octal), so there are a total of 1,048,576 individually addressable positions on the display screen. You can address any of these positions by identifying the appropriate x- and y-coordinates associated with the desired position. The distance between one point and the next in the default display coordinate system is one unit known as a raster. The software uses single raster units as a default condition, but you can establish other coordinate systems as desired.

The viewing area capacity, when normal-size characters are produced, is 73 characters per line and 31 lines per screen.

The total drawing area available on the VS60 system extends from a lower left corner whose coordinate position is (x=-4095, y=-4095) to an upper right corner at (x=4095, y=4095). There are 8192 individual points on each axis, so this total logical screen consists of 67,108,864 individually addressable positions.

Figure 1-8 illustrates the relationship of the DECgraphic-11 viewing and drawing areas on the VS60.

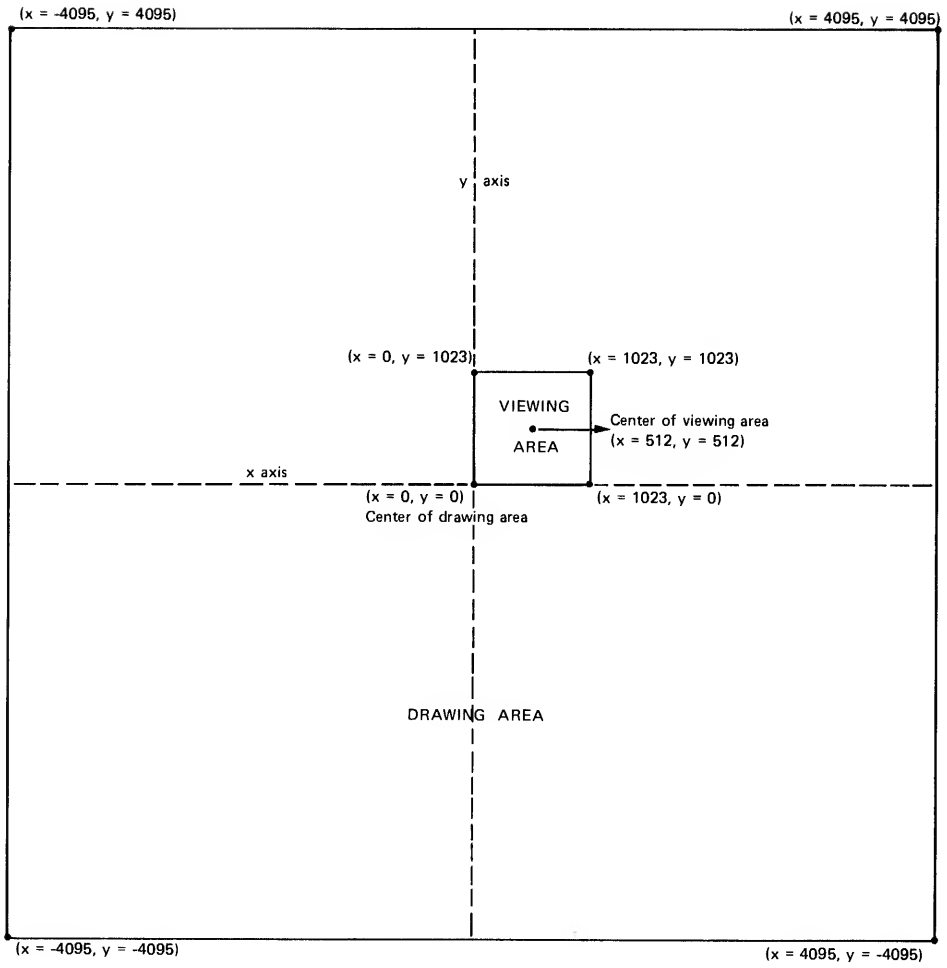

Figure 1-8 VS60 Display Screen

In defining points, lines, and other elements on the VS60 display screen, your coordinate specifications may normally extend beyond the bounds of the viewing area. Any display that extends beyond the bounds of the viewing area will appear to be truncated at the edge of the viewing area, but will actually extend into the remaining drawing area. However, if you specify an x or y value that exceeds -4095 or 4095, the display will be truncated or clipped at the edge of the drawing area.

In addition to the main area, the VS60 display processor also addresses a separate "menu" area. The main area is the 12-inch by 12-inch screen discussed above, and the menu area is a logical strip down the right side of the screen. It extends 1 1/2 inches (128 raster units) in the horizontal direction and 12 inches in the vertical direction. This 4-centimeter by 30-centimeter area accommodates 14 normal character positions. The menu area may be used for any purpose but is usually most helpful in displaying a list of options known as a menu, from which the user of your application program can specify a choice. An example of such a function is shown in Figure 1-9.

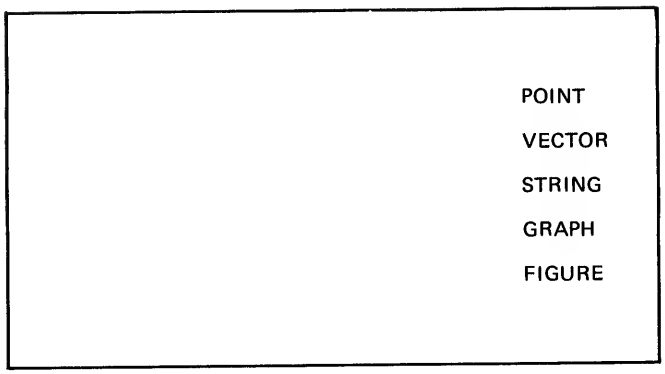

Figure 1-9 Menu Area

The display types listed here represent the choices available to the program user. He presses the light pen in the area of the VECTOR label, for example, to indicate that he wants a vector to be displayed by the program.

You can construct a menu by invoking the MENU subroutine (see Section 2.3.8). This subroutine allows you to specify the character strings to be displayed as menu entries, to assign a name or tag to each of these entries, and to specify the spacing of the entries in the menu area.

You may position a menu area at any location on the screen by including in the MENU call the coordinate positions at which the menu will begin. If these coordinates are not specified for the VS60, the menu is constructed in the separate logical menu area to the right of the main drawing area. On the VT11, the menu will begin at the left edge of the viewing area.

1.3.2 The Display File

When you issue subroutine calls and supply data used in performing graphics operations, the instructions and data for the display processor are stored in a special area of PDP-11 memory called the display file. You allocate this area of memory by defining a COMMON block called DFILE in the FORTRAN program that uses the graphics subroutines. The display file is initialized for use in DECgraphic-11 by means of the INIT subroutine (see Section 2.1.1).

The memory area used as the display file should be large enough to accommodate the display instructions and data required for the most complex image you intend to output during the current graphics session.

In displaying graphics on the screen, the VS60 or VT11 display processing unit (DPU) interacts with the display file much as the PDP-11 central processing unit interacts with PDP-11 memory. Just as the PDP-11 CPU accesses successive program instructions and data from memory, the VS60 or VT11 DPU accesses display instructions and data from the display file, independent of the operations of the PDP-11 CPU. The internal configuration is shown in Figure 1-10.

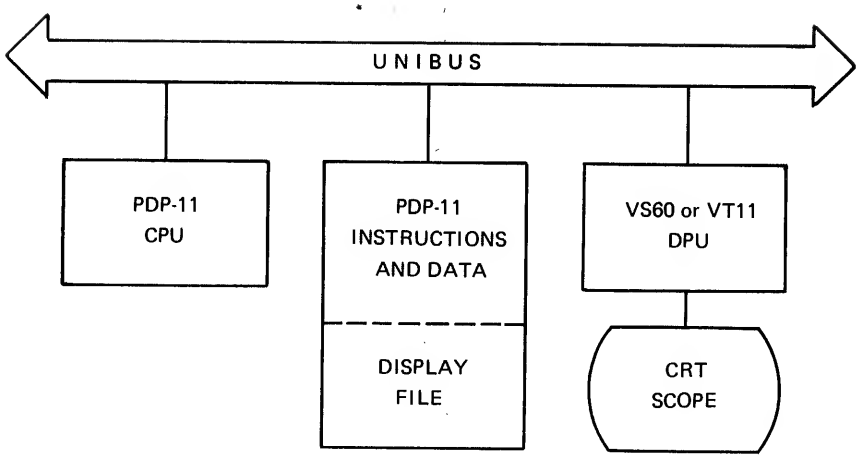

Figure 1-10 CPU and DPU

In order to create a display image, the display processor simply sequences through the display file to display images on the screen. The DECgraphic-11 system is self-synchronizing, that is, after completing one display file cycle, it jumps automatically back to the beginning of the file and begins to execute display instructions again. This facilitates the display of seemingly continuous images on the display screen.

As you issue subroutine calls and provide data used to plot graphs or display points on the screen, the display file fills up with instructions and data. Although you can erase elements in the display file, the space occupied by these elements cannot be reused until you issue a call to the CMPRS subroutine (see Section 2.9.1), which removes erased display elements and reclaims the display file space.

Many of the FORTRAN-callable subroutines described in this manual may be used to manipulate elements of the display file. By referencing pointers to elements stored in the file, you can delete, insert, or change display information. Section 1.3.5 describes the use of display file pointers in the DECgraphic-11 package.

There are a number of status parameters associated with the display file that you may set in order to change the display intensity level or line type, to cause elements to be light pen-sensitive, or to enable the blinking feature on a particular area of the screen. When you first initialize the display file, these parameters are set to certain initial values, as shown below.

- Light pen sensitivity (l in the subroutines) is disabled.
- Intensity level (i) is set to 4 in a range of 1 (faintest) to 8 (brightest).
- Flash (f) or blink mode is disabled.
- Line type (t) is set to solid.

There are a number of other default conditions associated with an intialized display file. These are described in the discussion of the INIT subroutine.

It is possible to save a display file as a data file on a mass-storage device such as disk, DECtape, or floppy disk. By saving and subsequently restoring a display file used to output a displayed image, you can cause a screen image to be displayed without having to rerun the FORTRAN program that was used to create the image. The SAVE and RSTR subroutines (see Sections 2.9.2 and 2.9.3) are used to back up and restore DECgraphic-11 display files.

1.3.3 Display Primitives

You may display a variety of different kinds of graphics elements on the screen, including points, line segments, and characters. These individual elements may be drawn by means of simple calls to FORTRAN subroutines and are known as display primitives. In drawing primitive elements, the position of the DECgraphic-11 display beam is relevant. The beam is the stream of electrons that is directed to a position on the display screen. At the time you initialize the graphics system (see the INIT subroutine, Section 2.1.1), the beam is positioned at the lower left corner of the screen (x=0,y=0). As primitive display elements are drawn at coordinate positions on the display screen, the beam changes position.

The following list summarizes the display primitives that may be specified in the DECgraphic-11 package:

- An absolute point, specified in absolute coordinate positions (e.g., (x=512, y=512)) on the screen (the APNT subroutine, see Section 2.3.1).
- A relative point at a relative coordinate position defined as the current beam position plus the (x,y) coordinate specification (RPNT, see Section 2.3.2).
- A vector that extends from the current beam position to an absolute coordinate position on the display screen (AVECT, see Section 2.3.4) (VS60 only).
- A vector that extends from the current beam position to a point on the screen relative to the beam position; the vector may be drawn in short or long vector format (VECT, SVECT, and LVECT, see Sections 2.3.3, 2.3.5, and 2.3.6).
- A character string and associated control codes to indicate extended characters, italic mode, rotation mode, subscripts, and superscripts (TEXT, see Section 2.3.7).

Primitive elements are usually combined on the screen into more complex graphics displays. In the display file, the subroutine calls used to create the primitives that are combined are often grouped together and stored as a named entity called a subpicture. Section 1.3.4 introduces the definition and use of subpictures, and Section 2.5 describes a variety of DECgraphic-11 subpicture-manipulation routines.

You may combine any kind or number of primitive elements into a subpicture. You may also create special types of subpictures called "graphs", "figures", and "numbers" by issuing simple subroutine calls to XGRA and YGRA (see Sections 2.5.1 and 2.5.2), FIGR (see Section 2.5.3), and NMBR (see Section 2.4.7).

All of the primitive elements defined by means of FORTRAN calls are stored in the display file and cause corresponding images to be displayed on the screen. It is possible to add, delete, and change individual primitives in the display file by making use of special DECgraphic-11 software pointers. You can create a pointer that addresses a particular primitive in the display file, advance it through the display file, primitive by primitive, and change or delete information in the referenced primitive. Section 2.7 describes the FORTRAN subroutines used to manipulate pointers and change primitive elements.

1.3.4 Subpicture Definitions

The DECgraphic-11 package supports a special subpicture facility that allows you to combine individual graphics elements into a structure known as a subpicture. Because many displayed pictures contain repeated images, it is often more efficient to define such images as subpictures; a subpicture can then be invoked when it is needed, saving you from having to repeat an almost identical sequence of display information. All subpictures defined in the system are assigned names or tags that can subsequently be referenced. A tag must be a positive integer in range 1 through 32767.

A subpicture can be defined, displayed, copied, erased, turned on and off, and altered. Subpictures are functionally very similar to subroutines. Calls to existing subpictures may be issued and nested to a depth of eight calls, to allow complex structures to be built up from simpler components.

A large number of the FORTRAN graphics routines described in Chapter 2 are oriented to the use of subpictures. The following are some of the most important subpicture operations you may perform in the DECgraphic-11 system:

- Defining the start of a new subpicture description or referencing an existing subpicture to be logically copied and renamed; subpicture definitions can be nested to a depth of eight calls (the SUBP subroutine, see Section 2.4.1)

- Terminating a subpicture definition and optionally specifying that any display file parameters (e.g., light pen sensitivity, intensity, flash mode, line type) that were reset for use in the subpicture are to be restored to the values they had before the subpicture was called (ESUB, see Section 2.4.2)

- Copying a subpicture and assigning a new tag to the copied subpicture (COPY, see Section 2.4.3)

- Turning a subpicture off temporarily and turning it on again, often to display an image all at once on the screen after it has been totally constructed in the display file (OFF and ON, see Sections 2.4.4 and 2.4.5)

- Erasing a subpicture definition from the display file (ERAS, see Section 2.4.6)

- Creating special subpictures called "numbers", "graphs", and "figures" (NMBR, XGRA, YGRA, and FIGR, see Sections 2.4.7, 2.5.1, 2.5.2, and 2.5.3 respectively)

- On the VS60, changing the size of characters and/or vectors to be displayed in a particular subpicture (CVSCAL, see Section 2.4.8)

- Changing individual primitives in a subpicture definition by advancing a pointer through that subpicture's portion of the display file (see Sections 1.3.5 and 2.6)

In addition to the descriptions of FORTRAN routines that define and access subpictures, a number of advanced programming techniques used in building subpictures are included in Section 3.1.

1.3.5 Display File Pointers

It is possible to insert, delete, or change individual primitives at any location in the display file. To access a primitive, you must reference it by moving a pointer to the appropriate display file location. There are 21 available pointers in the DECgraphic-11 package. These pointers are numbered 1 through 21 inclusive. In referencing display file elements, you can move all but the 21st pointer, which serves as the system pointer and should not be referenced in a user program.

The availability of 20 individual pointers enables you to keep track of as many as 20 different primitive locations. To alter a simple primitive, you must first determine the location of the pointer you will be using, and then advance the pointer until it references the particular primitive you wish to change.

The following list summarizes the major operations that can be performed by manipulating pointers:

- Setting a pointer at a particular primitive within a subpicture (the POINTR subroutine, see Section 2.6.1)

- Advancing a pointer by a certain number of primitive elements (ADVANC, see Section 2.6.2)

- Returning the coordinate positions of the primitive currently referenced by a pointer (GET, see Section 2.6.3)

- Changing the coordinate values of a primitive referenced by a pointer referenced by a pointer (CHANGE, CHANGA, and CHANGT, see Sections 2.6.4 through 2.6.6)

- Inserting a new element in the display file just before the primitive referenced by a pointer (INSERT, see Section 2.6.7)

- Erasing the primitive referenced by a pointer (ERASP, see Section 2.6.8)

1.3.6 User Coordinate Systems And Windows

As described in Section 1.3.1, the standard DECgraphic-11 display screen viewing area is defined as the area whose lower left corner has coordinate positions (x=0, y=0) and whose upper right corner has coordinate positions (x=1023, y=1023). In this viewing area, each axis has 1024 individually addressable points. The coordinate system that describes this area is unit-scaled, that is, the distance from one point to another is one raster unit, and consecutive points on an axis are addressed in increments of one. For example, the first three points along the x axis are (x=0, y=0), (x=1, y=0), (x=2, y=0).

It is possible to change this default coordinate system to almost any system that you find convenient. For example, you might want to address consecutive rasters in increments of ten units. The first three points along the y axis might then be (x=0, y=0), (x=0, y=10), (x=0, y=20). The physical space between these points would remain one raster unit, but you would describe the positions according to your own coordinate system. The relationship between the standard unit-scaled coordinate system and your own coordinate system (raster units and user units) is called the scaling factor. In the case described, the scaling factor along both axes is 10.

The SCAL subroutine (see Section 2.2.3) is used to define the new coordinate system by identifying the coordinate positions of the lower left and upper right corners of the display screen viewing area. You may optionally specify that the scaling factors along the x and y axes are to be returned. To restore the default coordinate system, you must issue a call to the NOSC subroutine.

As mentioned in Section 1.3.1 in the description of the VS60 display screen, the screen viewing area can actually be considered a window on a larger image-definition or drawing area. On the VS60, scaling the viewing area also scales the drawing area. The specification:

```
CALL SCAL (0.,0.,1.,1.)
```

scales the viewing area to user coordinate positions (x=0,y=0) at the lower left corner and (x=1,y=1) at the upper right corner. This specification also scales the total drawing area to extend from (x=-4,y=-4) to (x=4,y=4).

You may freely display pictures in any portion of the total VS60 drawing area. In order to examine a display that is outside the standard viewing area, however, you must move the viewing window to the relevant portion of the drawing area. You do this by redefining the viewing window. The WINDOW subroutine (see Section 2.2.5) is used to perform this redefinition. In it, you specify the lower left corner of the new viewing window, for example:

```
CALL WINDOW (-4095.,-4095.)
```

Because the viewing area is always a standard physical size (1024 x 1024 raster units), the upper right corner of the new viewing window is thus (x=-4095 + 1024, y=-4095 + 1024) or (-3071, -3071). The following diagram illustrates the placement of the new viewing window in the total drawing area.

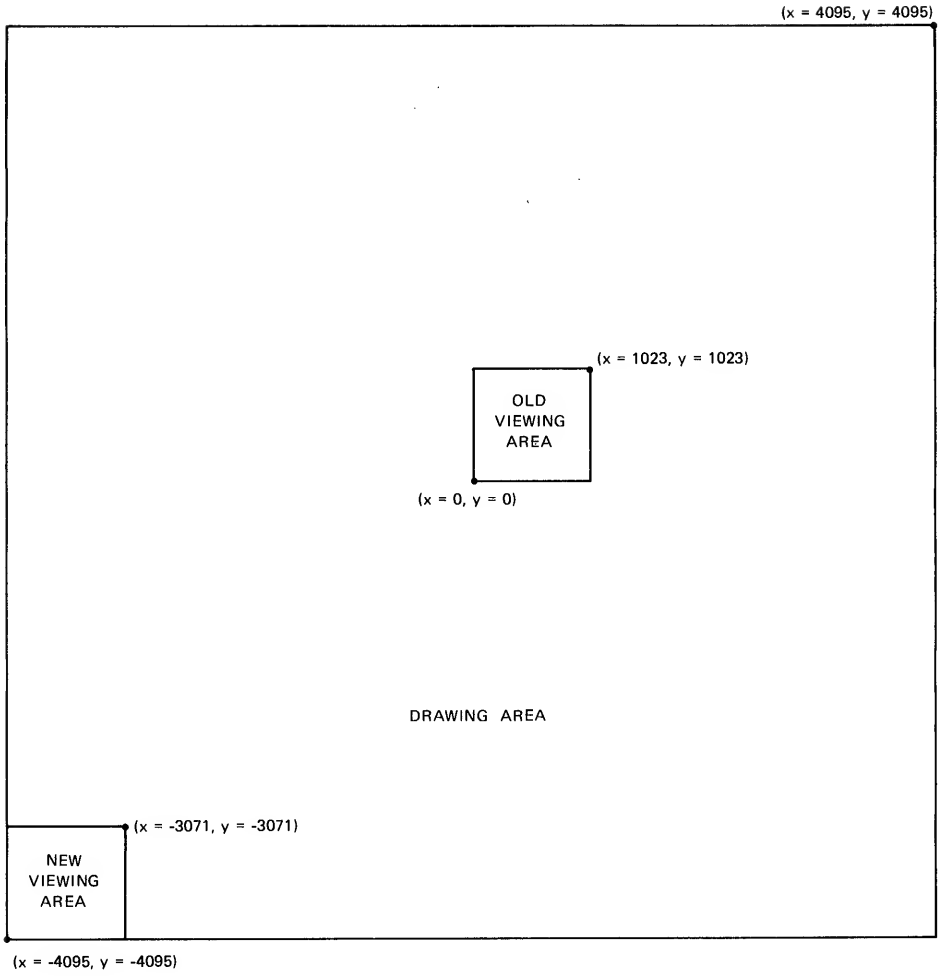

Figure 1-11 Viewing Window

1.3.7 Light Pen Interaction and Tracking

DECgraphic-11 is an interactive graphics system. It allows you to output a wide variety of graphics elements on the display screen, to change these elements dynamically, and to interact with the system. To facilitate this user interaction, the system supports a light pen, a solid-state light-detecting device that can be pointed at any display element on the screen. It consists of a photosensitive diode. If a graphics element has been made light pen-sensitive, a PDP-11 interrupt will occur when the beam is traced in front of the light pen. This is known as a "light pen hit". The display processor hardware retains information on the characteristics and coordinate positions of the hit.

The VS60 light pen also has a tip switch, whose status (in or out) is set if you press or remove the tip from the screen to initiate certain kinds of interaction. For example, the tip switch may be used to confirm detection of a particular graphics element or menu item. The light pen's tip switch has its own separate PDP-11 interrupt and

allows you to exercise additional control in interacting with the display. For example, you might perform certain actions after a light pen hit with the tip switch set, and alternative actions after a light pen hit without the tip switch set.

The light pen is often used to select an element from the menu area of the display screen. The application program may display a list of options, as shown in the following example:

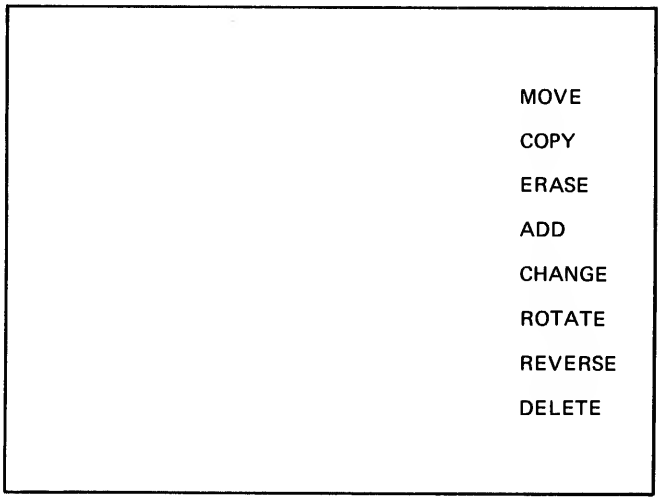

Figure 1-12 Light Buttons

Each of the labels displayed in the menu area of the screen is known as a light button. If you press the tip of the light pen on the screen in the area of the ADD element, for example, the appropriate information about the position of the light pen will be returned to the FORTRAN program and you can initiate a program branch to the appropriate FORTRAN statements.

The coordinates of a light pen hit will be returned with an accuracy of precisely one raster unit for the VT11 and approximately four raster units for the VS60.

The interactive nature of the DECgraphic-11 system is increased by support of a tracking object. The tracking object is a diamond-shaped image that can be displayed to keep track of light pen hits. Because it is internally stored as a subpicture of relative vectors, it can be moved freely around the screen to follow the light pen.

Figure 1-13 illustrates the shape of the tracking object.

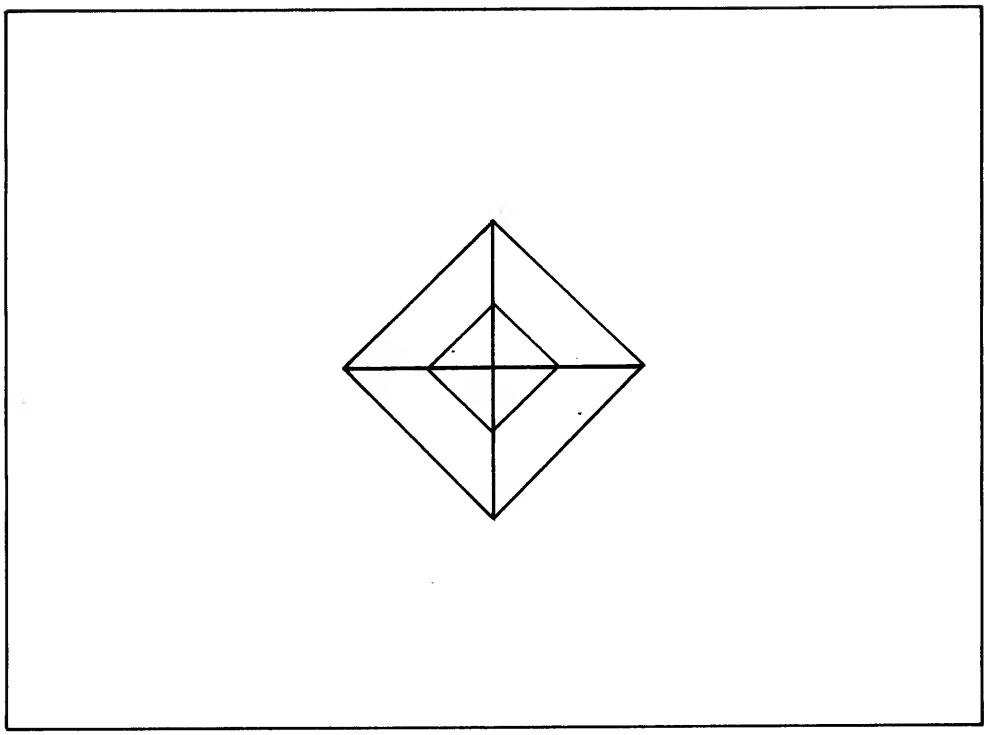

Figure 1-13 Tracking Object

It may initially be positioned at any location on the display screen. When a light pen hit takes place on the tracking object, the object centers itself automatically on the hit.

You may alter primitive elements such as points and vectors by attaching them to the tracking object, moving the object, and then detaching the primitives. You may also define a logical grid consisting of a set of imaginary points, evenly spaced at user-defined intervals on the display screen. The tracking object can then be automatically positioned at the point on the grid nearest the light pen hit. In this way, you can adjust the coordinate positions of a light pen hit.

The following list summarizes the major capabilities of the DECgraphic-11 FORTRAN support package in the area of light pen and tracking object utilization:

Testing for a light pen hit on the display screen and returning the following information for successful hits (the LPEN subroutine, see Section 2.8.1):

-scope on which the hit occurred (VS60)

-tag of the subpicture in which the hit occurred

-x and y coordinates of the hit

-number of the primitive within the subpicture

-array in which the subpicture precedents are stored

-screen area (main or menu) in which the hit occurred

-status of the tip switches (VS60)

- Positioning the tracking object at specified coordinate positions (TRAK, see Section 2.8.2)
- Returning the current coordinate position of the tracking object (TRAKXY, see Section 2.8.3)
- Changing primitives by attaching them to the tracking object, moving the tracking object by means of the light pen, and then detaching the primitives (ATTACH and DETACH, see Sections 2.8.4 and 2.8.5)
- Moving the tracking object to the nearest coordinate positions on the logical grid (GRID, see Section 2.8.6)

1.4 GRAPHICS SUBROUTINES

The user interface to the DECgraphic-11 interactive graphics package is a collection of graphics subroutines. You may invoke these routines by issuing simple subroutine CALLs from your FORTRAN programs.

The FORTRAN-callable graphics subroutines implemented for DECgraphic-11 support may be divided into a series of functional categories. Each of the following subsections summarizes the format and use of the subroutines in one of these categories. Each description includes the name, argument list, and effect of the graphics call.

Chapter 2 describes the graphics subroutines in more detail. Appendix A provides a summary of the FORTRAN calls and effects, arranged in alphabetic order for use as a reference.

Most of the graphics subroutines described below are identical for use in VT11 and VS60 graphics systems. Where differences exist between the subroutine calls on the two systems, an asterisk (*) is included before the subroutine name. Usually, the difference is simply that only one scope is supported on the VT11, so the s parameter is ignored in VT11 calls. In addition, there are a number of graphics routines that have been implemented to utilize the more powerful VS60 capabilities. These subroutines are distinguished by a cross (+) before the subroutine name. Calls to these routines are no-ops in VT11 systems.

Many of the subroutine calls summarized below contain optional parameters. If you want to include an optional parameter but do not want to specify all of the parameters that precede it, you must enter a comma for each omitted parameter. For example, to include an intensity (i) specification in the APNT call:

```
CALL APNT (x,y[,l,i,f,t])
```

you may specify the following:

```
CALL APNT (0.,512.,,-4)
```

The extra comma effectively holds a place for the omitted l parameter. Note that it is not necessary to specify or hold places for the trailing f and t parameters.

1.4.1 Initializing And Controlling The Display File

The subroutines in this category are used to initialize the display processor, allocate a memory area for use as the DECgraphic-11 display file, and control the use of the display file.

Call	Argument List	Effect
INIT	[(n)]	Clears the display screen, initializes the display file to use the first n words of the FORTRAN common block, DFILE, and sets the initial display file status parameters.
STOP		Halts the display processor, stops the display, and clears the display screen. The display may be restored by calling CONT.
CONT		Restarts the display processor, thus restoring the display interrupted by a call to STOP.
FREE		Disconnects the display file from the display processor, thus freeing the area of memory used by the file; this terminates graphics processing until the next call to INIT.

1.4.2 Setting Screen And Scaling Parameters

These routines set values for such parameters as display scope number, screen area, and user coordinate system.

Call	Argument List	Effect
+ SCOPE	(n)	Specifies that subsequent graphics calls refer to scope 1 (n=1) or 2 (n=2).
+ AREA	(n)	Specifies that subsequent graphics calls refer to the main viewing area (n=1) or the menu area (n=2).
SCAL	(x0,y0,x1,y1[,FX,FY])	Defines a new coordinate system in which (x0,y0) identifies the lower left corner of the user-specified screen, and (x1,y1) identifies the upper right corner of the screen; optionally returns the effective X and Y scaling factors in FX and FY.
NOSC		Restores the default unit-scaled coordinate system ((x0=0, y0=0) to (x1=1023, y1=1023)) for subsequent graphics calls, and eliminates any user-defined coordinate system.

Call	Argument List	Effect
+ WINDOW	(x,y)	Defines a window on the drawing area of the display screen by specifying the coordinate positions (x,y) of the lower left corner of the user-defined area.

1.4.3 Generating Graphics Primitives

The routines described below insert primitives or basic picture elements, such as points, vectors, and characters, in the display file. As each element is inserted in the file, it is also displayed immediately on the screen. Many of the routines in this category can also be used to change the basic display file status parameters:

- light pen enable; initial value is disabled
- intensity level (1-8); initial value is 4 (with normal adjustment, this makes the primitive bright enough to be detectable by a light pen)
- flash (blink) mode; initial value is off
- line type (solid, long-dashed, short-dashed, dot-dash); initial value is solid

A full description of the meaning of l, i, f, and t that can be specified in the routines summarized below is included in Section 2.3.

Call	Argument List	Effect
APNT	(x,y[,l,i,f,t])	Positions the beam at the absolute position represented by (x,y) and may display a dot at that position; optionally changes l, i, f, and t parameters.
RPNT	(x,y[,l,i,f,t])	Moves the beam from its current position to the relative position represented by (x,y) and may display a dot at that position. If the current position is (10,20) the beam is moved to (10+x, 20+y). Optionally changes l, i, f, and t parameters.
+ AVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the absolute point represented by (x,y); optionally changes l, i, f, and t parameters.
VECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position represented by (x,y), using the short vector format (see Section 2.3.5) when possible; optionally changes l, i, f, and t parameters.

Call	Argument List	Effect
LVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position represented by (x,y), using the long vector format (see Section 2.3.6); optionally changes l, i, f, and t parameters.
SVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position represented by (x,y), using the short vector format; optionally changes l, i, f, and t parameters.
TEXT	(a1[,a2...])	Displays the character strings supplied in the call, (a1, etc.) beginning at the current beam position. Characters specified in the call may include control codes indicating italic mode, character rotation, and superscript or subscript mode.
MENU	([x0],y0,dy,m[,'n1'[,'n2']...])	Displays a list of up to 10 items to be used as a menu. The items are the character strings n1, n2, and so on, and the first item is displayed at the coordinate position represented by (x0,y0). If x0 is omitted from a VS60 call, the items are displayed in the hardware menu area of the screen; on the VT11, they begin at the left edge of the viewing area. The dy parameter represents the vertical spacing between menu items. The m parameter represents the tag assigned to the first item or subpicture (n1); subsequent subpictures are tagged sequentially, so if there are 10 subpictures, the 10th subpicture's tag is m+10-1 or m+9.

1.4.4 Defining And Using Subpictures

The routines in this category generate subpictures and control the display and manipulation of subpictures on the screen.

Call	Argument List	Effect
SUBP	(m1[,m2])	Begins the definition of a subpicture whose tag is m1; all primitives specified in subsequent graphics calls are considered a part of the

	Call	Argument List	Effect
			subpicture, until the occurrence of a terminating call to ESUB. If the m2 parameter is included in the call, a new subpicture tag, m1, references an existing subpicture whose tag is m2.
*	ESUB	[(m)]	Terminates the definition of the current subpicture, created by means of the previous call to SUBP; if the m parameter is included in the call (VS60 only), the display processor status parameters (e.g., light pen enable, intensity) that were in effect before the current subpicture was invoked are restored.
	COPY	([m1],m2)	Creates a copy of the existing subpicture whose tag is m2 and assigns it the tag m1; if the m1 parameter is omitted, subpicture m2 is copied to the currently open subpicture.
	OFF	(m)	Temporarily turns off the subpicture whose tag is m.
	ON	(m)	Turns on subpicture m.
	ERAS	[(m)]	Erases the definition of subpicture m from the display file; if parameter m is omitted, the tracking object is removed from the screen.
	NMBR	(m,var[,n,format])	Creates a special numeric subpicture whose tag is m. This call formats the numeric contents of a FORTRAN variable, var, with a width of n in the specified format. If the format parameter is omitted, the default format of F16.8 is used.
+	CVSCAL	(m[,ifc,ifv])	Scales the size of displayed characters and vectors in subpicture m; ifc may be in range 1-4, where 1 is one-half normal character size, and 4 is twice normal size (normal is 2); ifv must be in range 1-15, where 1 is one-fourth normal size and 15 is three and three-fourths normal size (normal is 4).

1.4.5 Displaying Graphs And Figures

These routines create special subpictures that display graphs and figures on the screen and return information about these pictures. Note that several of the routines may be used to alter the display status parameters (l,i,f,t) described in Section 2.3).

Call	Argument List	Effect
XGRA	(dy,A,n,m[,l,i,f,t])	Creates a special graph subpicture m, which consists of a series of points. The x values of the points to be plotted are specified in the first n elements of array A, and the y values are given by integral multiples of dy. Optionally changes l, i, f, and t parameters.
YGRA	(dx,A,n,m[,l,i,f,t])	Creates a special graph subpicture m, which consists of a series of points. The y values of the points to be plotted are specified in the first n elements of array A, and the x values are given by integral multiples of dx. Optionally changes l, i, f, and t parameters.
FIGR	(A,n,m[,l,i,f,t])	Creates a special figure subpicture m. The figure is plotted from the first n (x,y) coordinate increment pairs specified in array A. Optionally changes l, i, f, and t parameters.
AGET	(m,j,Z)	Returns in variable Z the jth primitive of the graph or figure subpicture whose tag is m.
APUT	(m,j,b)	Assigns value b to the jth primitive of the graph or figure subpicture whose tag is m.
FPUT	(m,j,b)	Assigns value b to the jth primitive of the figure subpicture whose tag is m; adjusts the next element of the subpicture to insure that subsequent points in the figure will be at the same absolute coordinate positions on the screen.

1.4.6 Using Display File Pointers

The routines in this category manipulate any of the 20 user pointers in the system and change the contents of the primitives in the display file referenced by these pointers.

Call	Argument List	Effect
POINTR	(k,m[,j])	Sets pointer k to reference the jth primitive of the subpicture whose tag is m; if j is omitted, the pointer will be set to the first primitive of the subpicture. The value of k must be in range 1-20.
ADVANC	(k[,n])	Advances pointer k (1-20) by n primitives from its current position; if n is omitted, advances by one to the next primitive in the display file.
GET	(k,X,Y)	Returns in (X,Y) the coordinate positions of the primitive referenced by pointer k (1-20).
CHANGE	(k,x,y)	Changes the primitive referenced by pointer k (1-20) to the new value specified in (x,y). This routine can be used for primitives defined by the routines AVECT, VECT, SVECT, LVECT, RPNT, APNT, XGRA, and YGRA.
CHANGA	(k,x,y)	Changes the primitive referenced by pointer k (1-20) to the new value specified in (x,y). Like CHANGE, this routine can be used for primitives defined by the routines AVECT, VECT, SVECT, LVECT, RPNT, APNT, XGRA, and YGRA, but for VECT, SVECT, LVECT, and RPNT, subsequent graphics elements are adjusted to be displayed at the same absolute screen position.
CHANGT	(k,a1[,a2...])	Changes the character string primitive referenced by pointer k (1-20) to the new string supplied in the call (a1, etc). Characters specified in the call may include control codes indicating italic mode, character rotation, and superscript or subscript mode.
INSERT	[(k)]	Reopens the display file for insertion of primitives (specified in subsequent graphics calls). Insertion begins before the primitive referenced by pointer k (1-20). If k is omitted, the insert operation is terminated.
ERASP	(k)	Erases the primitive referenced by pointer k (1-20), and repositions the pointer at the next primitive in the display file.

1.4.7 Altering Display File Status Parameters

The routines described below are used to reset explicitly the following display file status parameters:

- light pen enable
- intensity level
- flash mode
- line type

These parameters can also be changed in conjunction with subpicture definition or image generation in a variety of other graphics calls (see Sections 1.4.3 and 1.4.5). The meaning of the specified values of l, i, f, and t in the following calls is explained in Section 2.3.

Call	Argument List	Effect
SENSE	(k[,l,s])	Depending on the value of l, enables or disables light pen sensitivity for the primitive referenced by k (1-20) in the display file associated with scope s (1 or 2).
INTENS	(k[,i])	Changes the intensity level of the primitive referenced by k (1-20) to the brightness specified in i (1-8) and/or intensifies the referenced primitive.
FLASH	(k[,f])	Depending on the value of f, enables or disables flash or blink mode for the primitive referenced by k (1-20).
LINTYP	(k[,t])	If t is a legal line type value (1-4), changes the line type of the primitive referenced by k (1-20) to the type specified in t.

1.4.8 Facilitating Light Pen Interaction

The routines described below handle interactions with the light pen and tracking object.

Call	Argument List	Effect
LPEN	(IH,IT[,X,Y,IP,IA,IM,IT1,IT2])	Indicates whether or not a light pen hit has taken place, and returns information about the hit in the following variables: IH: nonzero if light pen hit has occurred (1 for scope 1, 2 for scope 2); always 0 or 1 for the VT11.

Call Argument List	Effect
	IT: tag of the subpicture in which the hit occurred.
	X,Y: coordinates of the hit.
	IP: number of the primitive within the subpicture at which the hit occurred; undefined if the primitive is not in a subpicture.
	IA: array in which the precedents or ancestors of subpicture IT are stored (subpicture tags in order, starting with innermost nested subpicture).
	IM: screen area of light pen hit (1 for main area, 2 for menu area); always 1 for the VT11.
	IT1,IT2: status of the light pen tip switches for scopes 1 (IT1) and 2 (IT2) (0 for off, 1 for on); for the VT11, IT1 is always 1 and IT2 is always 0.
* TRAK (x,y[,s])	Positions a tracking object on the screen of scope s at coordinate position (x,y) and centers the object on any light pen hit within the tracking area.
* TRAKXY (X,Y[,s])	Returns the coordinates of the current position of the tracking object in (X,Y) for scope s.
* ATTACH (k[,n,s])	Attaches the primitive referenced by k (1-20) to the tracking object on scope s. When the primitive is a long vector, it will be attached to the object and will follow it; if n is positive or omitted, the vector's origin is stationary and the destination end will move; if n is negative, the destination end is stationary and the origin end will move. If the primitive is an absolute point or vector, n need not be specified.

Call	Argument List	Effect
* DETACH	[(s)]	Detaches all primitives from the tracking object on scope s.
* GRID	(gx,gy[,s])	Moves the tracking object on scope s to the nearest point on the grid and automatically detaches any attached primitives. Parameters gx and gy define the spacing of points on the grid.

1.4.9 Performing Display File Utility Functions

The routines in this category are used to compress, save, and restore the display file.

Call	Argument List	Effect
CMPRS		Compresses the display file by removing all erased primitives and subpictures and reclaiming the space used by them.
SAVE	('[dev:]file[.ext]')	Saves the display file by writing it onto the mass-storage file identified in the call.
RSTR	('[dev:]file[.ext]')	Restores in the memory area of the current display file the display file stored on the mass-storage file identified in the call.

1.4.10 Performing Advanced Display File Functions

The routines described below provide advanced display file manipulation facilities. They should be used only by experienced DECgraphic-11 users.

Call	Argument List	Effect
DPTR	(I)	Returns in I the position of the display file array element in which the next word entered in the display file will be stored.
DPYNOP	(n)	Inserts n display no-op instructions at the current position in the display file.
DPYWD	(i,j)	Inserts the 16-bit data word (i) in the display file at the current position in the display file; displays the word on the screen if the value of j is zero.

CHAPTER 2

DECGRAPHIC-11 GRAPHICS SUBROUTINES

This chapter describes in detail the graphics subroutines implemented for use in the DECgraphic-11 FORTRAN support package. You can invoke any of the subroutines discussed here by issuing simple subroutine calls from your FORTRAN programs. Each of the sections in this chapter describes the use of the subroutines in one of the 10 major functional categories listed below:

- initializing and controlling the display file
- setting screen and scaling parameters
- generating graphics primitives
- defining and using subpictures
- displaying graphs and figures
- using display file pointers
- altering display file status parameters
- facilitating light pen interaction
- performing display file utility functions
- performing advanced display file functions

2.1 INITIALIZING AND CONTROLLING THE DISPLAY FILE

This section describes the subroutines used to allocate memory for the DECgraphic-11 display file, initialize the display file, and control access to it.

2.1.1 INIT: Initializing the Display File

Form: CALL INIT [(n)]

The INIT subroutine sets up a display file for use in performing graphics operations or reinitializes the file for subsequent use. INIT clears the display screen, sets a variety of display parameters to initial values, and allocates an area of memory for use as the display file. The display file is used to store the instructions and data to be accessed by the display processor and is accessed by most of the graphics routines described in this chapter.

When you begin a graphics session, you must issue a call to the INIT subroutine in the form:

```
CALL INIT (n)
```

where n is the number of words to be allocated to the display file.

Before issuing the call, you must allocate an area of memory for use by the display file by defining a FORTRAN common block called DFILE. The amount of memory allocated for display file use must be sufficient to accommodate the largest display file that you anticipate creating during the current graphics session. If you exceed the size of display file, an error message will be returned by the system:

```
DISPLAY FILE FULL
```

and program execution will terminate.

The following example illustrates the creation of a FORTRAN COMMON block and the use of a call to INIT.

```
COMMON/DFILE/I(400)
CALL INIT (400)
```

This COMMON specification allocates 400 words of memory to the area named DFILE. When you first call INIT, you must specify the amount of the DFILE area to be used as the display file in the subroutine call. Subsequent calls to INIT need not include this parameter.

At any time during the graphics session, you can clear the screen and reinitialize the display file by issuing an INIT call of the following form:

```
CALL INIT
```

The INIT routine must be invoked in order to perform any of the graphics operations described in this chapter. Once the display file has been created, subsequent calls to image-generation subroutines cause the display processor to add new instructions and data to the display file or to modify the existing contents of the file. Any FORTRAN program that has called INIT will not return to the monitor immediately after execution or error detection. Instead, the system will display the following message on the console:

```
TYPE <CR> TO EXIT
```

Regardless of whether or not an error has occurred, you will be able to view the display created by the program without having to issue a READ or PAUSE instruction.

NOTE

Do not use the FORTRAN library routine, USEREX, after a call to INIT. The INIT subroutine issues its own call to USEREX.

The INIT subroutine establishes initial values for a variety of display file status parameters. The conditions listed below are in effect after an initial call to INIT:

1. The standard coordinate system is enabled.
2. The beam is positioned at the lower left corner of the viewable area of the screen (x=0, y=0).
3. Light pen interaction is disabled.
4. The intensity level is set to 4.
5. Flash (blink) mode is turned off.
6. Solid line type is established.
7. Characters are displayed in the normal font, not in italics or shift-out mode.
8. For the VS60, scope 1 is on and scope 2 is off.
9. On the VS60, any display is directed to the main viewing area, not the menu area.

Example:

This example initializes the DECgraphic-11 display processor and allocates a 2000-word display file.

```
COMMON/DFILE/IBUF(2000)
    .
    .
    .
CALL INIT (2000)
    .
    .
    .
```

2.1.2 STOP: Stopping the Display

Form: CALL STOP

The STOP subroutine halts the use of the display processor, stops the display of the current graphics, and clears the display screen. The stopped display may be restored by a call to the CONT subroutine (see Section 2.1.3). On the VS60, STOP clears the screens on both scopes, if both are enabled (see Section 2.2.1). STOP halts the transmission of interrupts from the VT11 or VS60 display processor. Because interrupts from the display processor are not being processed, the CPU executes at a significantly faster rate. Stopping an already stopped display has no effect.

2.1.3 CONT: Restoring the Display

Form: CALL CONT

The CONT subroutine is used to restore the display that was interrupted by a call to STOP. This routine causes the display processor to be restarted and causes interrupts from the VT11 or VS60 to be transmitted once again.

2.1.4 FREE: Clearing the Display File Area

Form: CALL FREE

The FREE subroutine is issued when you need to use the memory area allocated the display file for another purpose. A call to FREE effectively disconnects the display file from the display processor, thus freeing the space used by the display file. FREE also has the effect of clearing the display screen.

After you have issued a call to the FREE subroutine, a display file is not available for use in graphics processing. If you subsequently require access to the display file in your program, you must issue another call to INIT. The DFILE COMMON block definition remains in effect.

2.2 SETTING SCREEN AND SCALING PARAMETERS

The subroutines described in this section are used to select scope and screen area values and to establish your own coordinate system for use in subsequent graphics displays.

2.2.1 SCOPE: Selecting a Display Scope

Form: CALL SCOPE (n)

The SCOPE subroutine is used in VS60 display systems to enable or disable either of the two scopes supported by DECgraphic-11. A call to SCOPE has no effect on a VT11 system. After you have enabled a scope, you are free to issue graphics calls that will display elements on the enabled device.

Legal forms of the SCOPE call are the following:

```
CALL SCOPE (1)
CALL SCOPE (2)

CALL SCOPE (-1)
CALL SCOPE (-2)
```

The first pair of calls enables scopes 1 and 2 respectively. The second pair disables scopes 1 and 2 respectively. If parameter n has any other value, the subroutine call is ignored.

An enabled scope remains enabled until you explicitly disable it, so both VS60 scopes may be used at the same time. However, if a call to INIT is made during program execution, the initial scope setting will be reestablished, and only scope 1 will be enabled.

An example of using SCOPE is included in the description of the AREA subroutine (see Section 2.2.2).

2.2.2 AREA: Selecting the Main or Menu Area

Form: CALL AREA (n)

On the VS60, the AREA subroutine allows you to switch subsequent graphics displays to one of the two distinct areas of the display

screen: the main viewing area or the menu area. A call to AREA has no effect on a VT11 system.

Legal forms of the AREA call are the following:

```
CALL AREA (1)
CALL AREA (2)
```

When parameter n is 1, as in the first call, the display is switched to the main viewing area. When n is 2, the display is switched to the menu area. If parameter n has any other value, the subroutine call is ignored.

When you issue a call to AREA, the current beam position is not automatically changed, so you must reposition the beam before displaying graphics on the screen. It is not possible to access both the main viewing area and the menu area at the same time.

Example:

This example draws four absolute points:

- one at the center of the main area on scope 2
- one at the lower left corner of the menu area on scope 2, with the menu area flashing
- one at the lower left corner of the menu area on scope 1
- one at the center of the main area on scope 1

```
.
.
.
CALL SCOPE (2)
CALL APNT (512.,512.)
CALL AREA (2)
CALL APNT (0.,0.,,,1)
CALL SCOPE (1)
CALL APNT (0.,0.,,,-1)
CALL AREA (1)
CALL APNT (512.,512.)
   .
   .
   .
```

2.2.3 SCAL: Defining a User Coordinate System

Form: CALL SCAL (x0,y0,x1,y1[,FX[,FY]]

The SCAL subroutine allows you to define your own display screen coordinate system. The default coordinate system describes the standard screen viewing area as the area extending from (x=0,y=0) to (x=1023,y=1023). The area has a physical size of 1024 raster units on each axis, and each axis contains 1024 individually addressable points. Consecutive coordinate positions on an axis are addressed in increments of 1 (e.g., x=0, x=1, x=2), so the coordinate system is described as unit-scaled.

The SCAL subroutine is used to change this coordinate system to almost any other scale. For example, you can specify a virtual screen containing only one fourth the number of addressable points on the standard scale:

```
CALL SCAL (0.,0.,512.,512.)
```

where (0.,0.) is the lower left corner and (512.,512.) is the logical upper right corner of the display screen. The physical screen size remains the same (1024 raster units along each axis), but consecutive coordinate positions are now two raster units apart, not one. The scale factor along each axis is two.

If you specify the following calls:

```
CALL SCAL (0.,0.,200.,200.)
CALL APNT (0.,0.)
CALL VECT (200.,200.)
```

the vector that is produced will extend on the main display diagonal, completely across the screen.

You may specify one or two additional parameters in the SCAL call. These parameters represent variables to which the scaling factors for the x and y axes are returned. If FX is included in the call, the x scaling factor will be returned as the value of the variable; if you also include FY, the y scaling factor will be returned as well.

Note that each new call to SCAL defines the coordinate units for subsequent graphics calls. Any previous scale settings are negated by a new call to SCAL. However, if vectors and points are drawn with one scale factor and later modified by means of GET, CHANGE, or some other subroutine after changing the scaling, the new scale factors will be used. This can lead to a great deal of confusion, so you should generally attempt to perform all work on a set of graphics elements with the same scaling.

NOTE

The length and coordinates specified in every graphics call must be within the viewing area of the current coordinate system (the default system or the user-defined system established by a call to SCAL). Any graphics call that attempts to position the beam beyond the edge of the viewing area is ignored by the system.

Because all vector displacements and point coordinates are represented as integers in the display file, errors may accumulate when several relative vectors or points are drawn consecutively. This happens when the user-specified displacement or point coordinate scales to a non-integer number of physical screen units.

An example of using SCAL is included at the end of the description of the NOSC subroutine (see Section 2.2.4).

2.2.4 NOSC: Restoring the Standard Coordinate System

Form: CALL NOSC

The NOSC subroutine is used to eliminate any user-defined display screen coordinate system that is in effect and to restore unit scaling for subsequent graphics calls. This reestablishes the standard coordinate system in which the screen is described as the area between coordinate positions (x=0,y=0) and (x=1023,y=1023).

Example:

This example changes the scaling factor, outlines the display screen with vectors, and then restores the default unit scaling.

```
      .
      .
      .
CALL SCAL (0.,0.,1.,1.)
CALL APNT (0.,0.,,-4)
CALL LVECT (1.,0.)
CALL LVECT (0.,1.)
CALL LVECT (-1.,0.)
CALL LVECT (0.,-1.)
CALL NOSC
      .
      .
      .
```

2.2.5 WINDOW: Establishing a Display Screen Window

Form: CALL WINDOW (x,y)

The WINDOW subroutine allows you to establish an effective window on the image-definition or drawing area of the VS60 display screen. With WINDOW, you can select the position of the drawing area you will view on your display screen. WINDOW is a no-op on the VT11.

The (x,y) parameters included in the WINDOW call represent the coordinate positions of the lower left corner of the user-defined window, where (x,y) must be in range (-4095.,-4095.) to (4095.,4095.). The size of the window is the size of the display screen viewing area, that is, 1024 by 1024 raster units. If the lower left corner is represented by (x,y), then the upper right corner is therefore (x+1023.,y+1023.).

In the following example:

```
CALL WINDOW (512.,512.)
```

the area of the screen shown in the diagram is defined as the window.

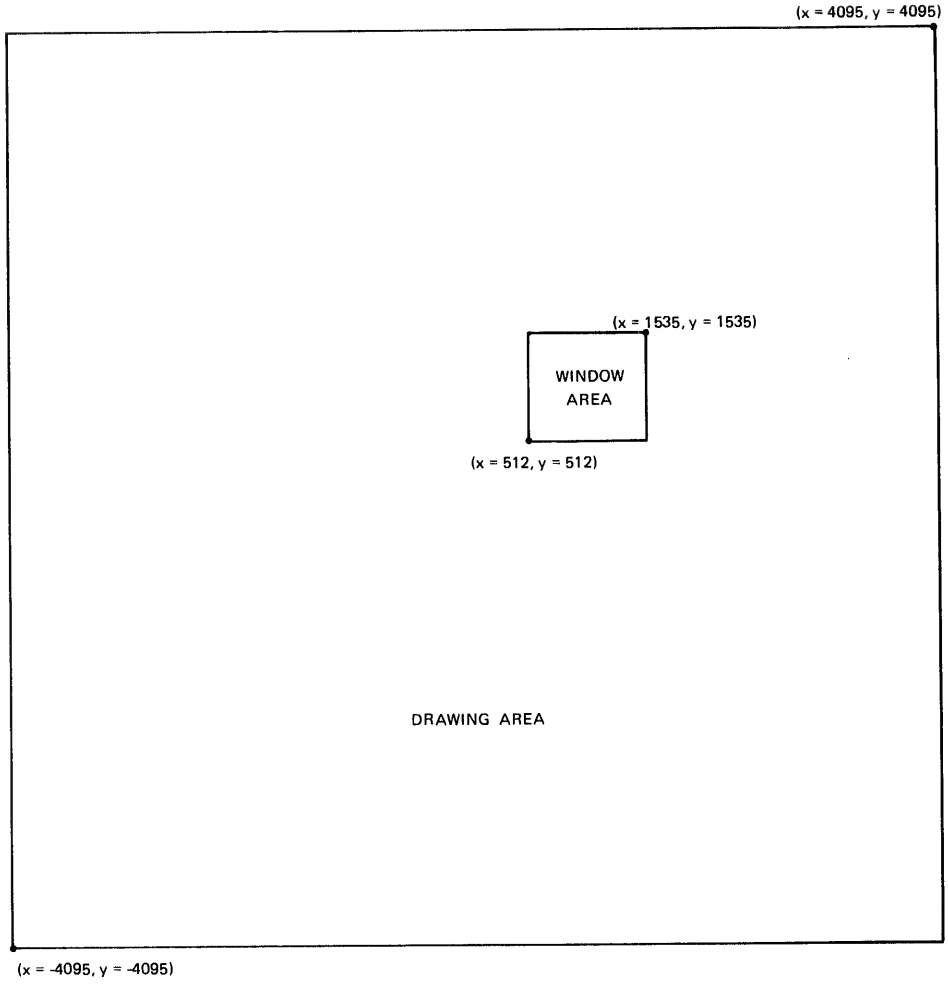

Figure 2-1 WINDOW Subroutine

Examples:

This example sets the viewing area of the screen to extend from a lower left corner of (1024.,0.) to an upper right corner of (2047.,1023.).

```
      .
      .
      .
CALL SUBP (1)
CALL WINDOW (1024.,0.)
CALL ESUB
      .
      .
      .
```

The viewing area is now changed to extend from (x,y) to (x+1023., y+1023.) Note that when you change a window by means of the CHANGE subroutine (see Section 2.6.4), the coordinates of the lower left corner are negated.

```
      .
      .
      .
CALL POINTR (10,1)
CALL CHANGE (1,-x,-y)
      .
      .
      .
```

2.3 GENERATING GRAPHICS PRIMITIVES

This section describes the FORTRAN subroutines used to generate graphics primitives such as points, vectors, and characters. These routines cause primitives to be inserted in the display file unless they are defined in a subpicture that is turned off (see Section 2.4.4). The primitives defined by means of these routines are also displayed immediately on the screen. Many of the routines in this category can be used to change the basic display status parameters summarized below. These parameters are set to the initial default values shown below when an INIT call is issued.

Argument	Meaning
l	Light pen enable Default: disabled l=positive. Light pen interaction is enabled, and a light pen interrupt will occur in subsequent graphics output when the light pen is pointed at an object on the display screen. l=0 or omitted. The value of the parameter does not change from its previous status. l=negative. Light pen interaction is disabled, and light pen interrupts will not occur.
i	Intensity level Default: 4; with normal screen intensity this level allows the light pen to detect the primitive. i=1 through 8. The intensify status is changed for current graphics output, and the intensity level (brightness of objects on the screen) is set to the value of i, where 1 is the faintest intensity and 8 is the brightest. i=0, omitted or greater than 8. The intensify status is changed for current graphics output, but the level does not change from its previous status for subsequent calls. i=-8 through -1. Current graphics output (except for characters) is not intensified, but the intensity level of subsequent graphics output is set to the absolute value of i. i=less than -8. Current graphics output is not intensified, nor is the intensity level changed for subsequent graphics output.
f	Flash mode Default: off f=positive. Current and subsequent graphics output is displayed in flash or blink mode. f=0 or omitted. The value of the parameter does not change from its previous status. f=negative. Flash mode is disabled for current and subsequent graphics output.

Argument	Meaning
t	Line type Default: solid t=1. Vectors in current and subsequent graphics output are displayed as solid lines. t=2. Vectors are displayed as long-dashed lines. t=3. Vectors are displayed as short-dashed lines. t=4. Vectors are displayed as dot-dash lines. t=0, omitted, negative, or greater than 4. The value of the parameter does not change from its previous status.

2.3.1 APNT: Displaying an Absolute Point

Form: CALL APNT (x,y[,l,i,f,t])

The APNT subroutine is used to position the beam at an absolute position on the display screen or to display a dot at that location. The point is represented by the coordinate position specified in parameters (x,y).

In the call to APNT, you may optionally specify new values for the display file status parameters l,i,f, and t, the meanings of which are described above. You can use the APNT subroutine to position the beam but not display a point by specifying a negative value for the intensity (i) parameter.

Example:

This example draws a flashing point in each corner of the display screen at intensity level 3.

```
      .
      .
      .
CALL APNT (0.,0.,,3,1)
CALL APNT (0.,1023.)
CALL APNT (1023.,0.)
CALL APNT (1023.,1023.)
      .
      .
      .
```

2.3.2 RPNT: Displaying a Relative Point

Form: CALL RPNT (x,y[,l,i,f,t])

The RPNT subroutine allows you to position the beam at a position on the display screen relative to the current beam position or to display a dot at that location. If the current beam position is represented by (x0,y0), then a call to RPNT will move the beam to the relative position represented by (x0+x,y0+y).

When a beam is repositioned by means of a call to RPNT, the displacement resulting from an (x,y) specification cannot exceed 63 raster units (approximately one-sixteenth of the full screen). The maximum allowable coordinate position that you should specify in (x,y) is thus (-63,,-63.) or (63.,63.). If your (x,y) specification exceeds 63 raster units, it will be truncated to 63 units by RPNT.

In the call to RPNT, you may optionally specify new values for the display file status parameters l,i,f, and t. As with APNT, you can use RPNT to position the beam but not display a dot by specifying a negative value for the i parameter.

Example:

This example draws a sine wave, made up of dots, across the display screen.

```
        .
        .
        .
     CALL APNT (0.,500.)
     Y0=500.
     DO 10 I=1,50
     Y=SIN(.125*I)*500.+500.
     CALL RPNT (20.,Y-Y0)
10   Y0=Y
        .
        .
        .
```

2.3.3 VECT: Drawing a Relative Vector

Form: CALL VECT (x,y[,l,i,f,t])

The VECT subroutine is used to draw a line segment or vector from the current beam position to a point on the display screen relative to the current beam position. If the current beam position is represented by (x0,y0) then a call to VECT will draw a line segment from (x0,y0) to (x0+x,y0+y).

The absolute values of the (x,y) specifications included in the VECT call must be less than (1024.,1024.) after any scaling. If the displacement of a vector after scaling is equal to (0.,0.), the vector will not be visible on the screen, regardless of the intensity level established by the current value of display parameter i.

The VECT subroutine uses the short vector format (one word for each vector) whenever possible in drawing line segments. If your program requires use of the long vector format, you should use LVECT (see Section 2.3.6).

In the call to VECT, you may optionally specify new values for the display file status parameters l,i,f, and t.

Example:

This example draws a triangle in the center of the screen.

```
      .
      .
      .
CALL APNT (500.,500.,,-4)
CALL VECT (25.,0.)
CALL VECT (0.,25.)
CALL VECT (-25.,-25.)
      .
      .
      .
```

2.3.4 AVECT: Drawing an Absolute Vector

Form: CALL AVECT (x,y[,l,i,f,t])

The AVECT subroutine draws a vector from the current beam position to an absolute point on the display screen. The point is represented by the coordinate position specified in parameters (x,y). AVECT is only used on the VS60 and is a no-op on the VT11. The AVECT subroutine uses the long vector format (two words per vector) in drawing line segments. In the call to AVECT, you may optionally specify new values for the display file status parameters l,i,f, and t.

Example:

This example outlines the screen with dot-dash lines (line type 4) at intensity level 4.

```
      .
      .
      .
CALL APNT (0.,0.,,-4,,4)
CALL AVECT (0.,1023.)
CALL AVECT (1023.,1023.)
CALL AVECT (1023.,0.)
CALL AVECT (0.,0.)
      .
      .
      .
```

2.3.5 SVECT: Drawing a Vector in Short Format

Form: CALL SVECT (x,y[,l,i,f,t])

The SVECT subroutine is used to draw a vector from the current beam position to a point on the display screen relative to the current beam position. If the current beam position is represented by (x0,y0), then a call to SVECT will draw a line segment from (x0,y0) to (x0+x,y0+y). The SVECT subroutine always uses the short vector format (one word per vector) in drawing line segments. The displacement resulting from an (x,y) specification cannot exceed 63 raster units. The maximum allowable coordinate position that can be specified in (x,y) is thus (-63,-63) or (63,63). If your displacement exceeds 63 raster units, it will be truncated to 63 units by SVECT. It is recommended that you use LVECT rather than SVECT if the line segment to be drawn may possibly exceed 63 raster units.

In the call to SVECT, you may optionally specify new values for the display file parameters l,i,f, and t.

2.3.6 LVECT: Drawing a Vector in Long Format

Form: CALL LVECT (x,y[,l,i,f,t])

The LVECT subroutine draws a vector from the current beam position to a point on the display screen relative to the current beam position. If the current beam position is represented by (x0,y0), then a call to LVECT will draw a line segment from (x0,y0) to (x0+x,y0+y). The LVECT subroutine always uses the long vector format (two words per vector) in drawing line segments. The displacement resulting from an (x,y) specification cannot exceed 1023 raster units. The maximum allowable coordinate position that can be specified in (x,y) is thus (-1023.,-1023.) or (1023.,1023.). It is recommended that you use LVECT rather than SVECT if the vector being drawn is of variable size and may exceed 63 raster units.

In the call to LVECT, you may optionally specify new values for the display file parameters l,i,f, and t.

2.3.7 TEXT: Displaying a Character String

Form: CALL TEXT (a1[,a2...])

The TEXT subroutine allows you to specify a character string including visible characters and control codes; this string describes characters to be displayed on the screen beginning at the current beam position. In the call to TEXT, you may include up to 10 arguments representing elements to be output. Each element to be output is represented by a parameter in the CALL format included above. A display element may include any of the following character types:

- normal character strings (a subset of ASCII printable characters)
- a count of carriage return/line feed pairs to be output
- special shift-out characters
- codes representing italic-font mode
- codes representing rotation mode
- codes representing subscripts or superscripts

A normal character string may consist of any number of printable characters with ASCII values greater than or equal to 40 (octal). These are the following:

- upper-case letters A through Z and lower-case letters a through z
- numbers 0 through 9
- a variety of special characters (see the examples in Section 1.1.3)

A normal string must be enclosed in single quotes within the TEXT call, as shown in the following example:

```
CALL TEXT ('ABCDEF')
```

If you want to specify a shift-out character or to indicate the use of italic mode or rotated text, you must include a special numeric value in the TEXT parameter list (a1[,a2...]). When the a argument is an integer in range -128 through -1, it is interpreted as a special code. If a is in range 1 through 31, it represents a count of the carriage return/line feed pairs to be output. Any other value of a will be interpreted as a text string.

The specific meanings of special TEXT codes are summarized below.

Value of a (when a is an integer)	Effect
1 to 31	Carriage return/line feed count
0	Ignored
-1	Shift-out character string follows
-2	Italic-mode character string follows
-4	Rotation-mode (90 degrees counter-clockwise) character string follows (VS60)
-8	Superscript in subsequent string (VS60)
-16	Leave superscript mode before displaying subsequent string (VS60)
-32	Subscript in subsequent string (VS60)
-64	Leave subscript mode before displaying subsequent string (VS60)

Options can be combined to represent selection of two or more modes; values are added to indicate that two or more options are being selected. For example, you would specify -3 to indicate an italic shift-out character string, and -7 to indicate a rotated italic shift-out character string.

A carriage return/line feed code in the TEXT call causes the next line of text to be output, starting at the left edge of the viewing area of the display screen. This will be either the main area or the menu area, depending on the AREA setting. You may specify a value of range 1 through 31 to indicate the number of carriage return/line feed pairs to be output and thus the number of lines to be skipped.

Shift-out characters consist of a set of 31 special characters, including Greek letters and mathematical symbols. These characters are invoked by specifying the special shift-out code (-1), followed by the ordinary character string that corresponds in sequence to the desired shift-out character string. This correspondence is indicated in the following list.

ERROR REPORT

The following page (p. 2-15) presents a table with an error. The shift-out character for "P" is a space (as the text mentions in paragraph 2). The table indicates "omega" as the shiftout character corresponding to "P."

From "omega/P" down, the shift-out characters should be one line lower, to make room for a "space"; the final character, "square," corresponds to "_" (underscore). The equivalent table in the DECgraphic-11 FORTRAN Programming Manual, p.2-34, is correct.

Shift-Out Character	Corresponding Character
λ	@
α	A
ϕ	B
Σ	C
δ	D
Δ	E
ι	F
γ	G
∩	H
ψ	I
÷	J
o	K
∴	L
μ	M
£	N
π	O
‖	P
Ω	Q
σ	R
τ	S
ε	T
←	U
→	V
↑	W
↓	X
Γ	Y
⊥	Z
≠	[
≈	\
∨	]
□	^

The following example causes the shift-out characters, $\alpha\phi\Sigma$, to be output:

```
CALL TEXT (-1,'ABC')
```

You may not include any character other than those included in the right-hand column above in a text string following a shift-out indicator. The letter O may be included to insert a blank in the shift-out text; this letter is converted to a space when output on the display screen.

When you are specifying subscripts or superscripts, you may include formats such as the following:

E^{-x^2+1}

To raise a power to a power, you must specify a superscript followed by a superscript, as in the following:

```
CALL TEXT ('E',-8,'-X',-8,'2')
```

Note that you remain in superscript mode, so the second -8 raises you to yet another level. To return to the normal line level to output '+1', you must include two -16 codes in the TEXT call; the following is the complete call.

```
CALL TEXT ('E',-8,'-X',-8,'2',-16,-16,'+1')
```

NOTE

Subscripts and superscripts are generated and positioned on the screen by means of an internal character scaling feature. Each level of subscript or superscript is displayed in a character size one level smaller than the size of the characters preceding the subscript or superscript. DECgraphic-11 supports four character sizes, and the normal character size is the second smallest of the four valid sizes (see the CVSCAL description, Section 2.4.8).

If you are displaying characters in the normal size, you may specify the inclusion of a single subscript or superscript. In this case, the normal characters have a scaling factor of 2, and the subscript or superscript has a scaling factor of 1. If you want to specify multiple levels of subscripts or superscripts, as in the example of TEXT above, you must begin with a character size larger than the normal size of 2. For example, if characters have a scaling factor of 3, then the characters at the first level of subscripting will have a scale of 2, and the characters at the second level will have a scale level of 1.

You may not allow the character scaling factor of the final subscript or superscript to fall below 1. If you specify too many subscript or superscript levels--for example, if you begin with a normal character size and include two or more successive subscripts or superscripts--the characters displayed after you return from subscript or superscript mode will have the wrong character size.

To output characters in italic mode, rotation mode, or as superscripts or subscripts, you must include the appropriate negative code, followed by the character string to be displayed in the desired mode.

You may position characters to be displayed by moving the beam to the desired coordinate position before issuing a call to TEXT. The lower left corner of the first character displayed will start at the current beam position. Every string displayed by means of a TEXT call must be terminated with a null byte. Compile-time string constants entered in subroutine calls satisfy this requirement, but you must remember to include this byte explicitly when generating your own string data in arrays.

You cannot ordinarily display "invisible" characters on the display screen. Text output is always visible regardless of the current value of the i parameter, unless an entire subpicture containing characters is turned off.

You may not actually specify new values for the display file parameters l,i,f, and t in the TEXT call, but you may issue a call to RPNT before the text call, as in the format shown below.

```
CALL RPNT (0.,0.,l,-i,f,t)
CALL TEXT ('string')
```

Note that a negative intensity level is specified here; this causes the beam to be positioned but does not display a point.

Examples:

This example displays labels and text illustrating the use of normal text, shift-out characters, italics, rotated text, superscripts, and subscripts.

```
      COMMON/DFILE/IBUF(1000)
      CALL INIT(1000)
      CALL SUBP(100)
      CALL APNT(0.,500.,,-4)
      CALL TEXT('ROTATED              ',
     X  -4,'ABCDEFGHIJKLMNOPQRSTUVWXYZ',1)
      CALL APNT(0.,450.,,-4)
      CALL TEXT('NORMAL               ',1)
      CALL TEXT('SHIFT OUT            ',
     X  -1,'ABCDEFGHIJKLMNOPQRSTUVWXYZ',1)
      CALL TEXT('ITALICS              ',
     X  -2,'ABCDEFGHIJKLMNOPQRSTUVWXYZ',1)
      CALL TEXT('SUPERSCRIPTED        ',
     X  '(X+Y)',-8,'Y',-8,'2',-16,-16,1)
      CALL TEXT('SUBSCRIPTED          ',
     X  'ARRAY',-32,'I',-32,'12',-64,-64,1)
      CALL TEXT('COMBINATIONS         ',
     X  'ROTATED/ITALICS',-6,'ABCD')
      CALL ESUB
      DO 100 J=1,4
      CALL CVSCAL(100,J)
      READ(5,10)I
10    FORMAT(I2)
100   CONTINUE
      STOP
      END
```

The following example draws the axes for a graph and labels them in italic-mode letters. Note that the label for the y axis is rotated.

```
      .
      .
      .
CALL APNT (100.,900.,,-4)
CALL LVECT (0.,-800.)
CALL LVECT (800.,0.)
CALL APNT (50.,200.,,-4)
CALL TEXT (-6,'POWER.LOSS')
CALL APNT (200.,50.,,-4)
CALL TEXT (-2,'FREQUENCY')
      .
      .
      .
```

2.3.8 MENU: Displaying Items in the Menu Area

Form: CALL MENU ([x0],y0,dy,m[,'n1'[,'n2']...])

The MENU subroutine allows you to establish a menu of elements on the display screen. As many as 10 items may be specified in a single call to MENU, and multiple calls may be issued. After a MENU operation is performed, the VS60 beam is automatically returned to the main area, so if multiple MENU calls are issued, they should be grouped together.

The particular area to be used for menu items may be selected by the user. If the x0 parameter is included in the call, then (x0,y0) represents the coordinate position at which the menu items begin. For example, if (x0,y0) is (512.,512.), then the first menu item will be displayed beginning at the center point on the viewable area of the display screen.

If x0 is omitted from the call, then the default hardware menu area is used on VS60 systems (see the diagram below), and y0 is used to identify the vertical placement of the first item in that menu area. On VT11 systems, the first menu item simply begins at the left edge of the viewing area. The following call invokes the effect shown below.

```
CALL MENU (,512.,-100.,56,'ERASE','COPY','MOVE')
```

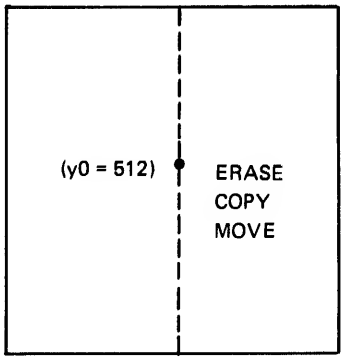

Figure 2-2 MENU Subroutine

The character strings of the items to be displayed in the menu are supplied in the 'n1'['n2']... parameters. In the example included above, the character strings 'ERASE', 'COPY', and 'MOVE' are the display items. The dy parameter represents the amount of vertical spacing between character strings displayed in the menu area. If the first item begins at (x0,y0) and there are six items, then the sixth item will begin at (x0,y0+(6-1)*dy). In the example included above, the first string is displayed starting at a y0 position of 512. Because dy is -100, the second item begins at (512+(2-1)*-100) or 412, and the third item at (512+(3-1)*-100) or 312.

The m parameter identifies the tag of the first item (n1) displayed in the menu area. In the example above, the tag representing character string 'ERASE' is 56. Successive character strings are tagged sequentially, so the second item, 'COPY', is 56+(2-1) or 57, and the third item, 'MOVE', is 56+(3-1) or 58. An example of using MENU is included in the description of the LPEN subroutine (see Section 2.8.1).

2.4 DEFINING AND USING SUBPICTURES

The subroutines described in this section are used to define subpictures for use in generating repeated images in graphics displays; to copy, erase, and move subpictures; to create special numeric subpictures; and to scale the size of characters and vectors used in subpictures. There is no intrinsic limit on the number of primitives that maybe defined in a subpicture.

2.4.1 SUBP: Defining a Subpicture

Form: CALL SUBP (m1[,m2])

The SUBP subroutine begins the definition of a new subpicture or references an existing subpicture. The m1 parameter represents the tag of the subpicture being defined; this tag can be referenced in subsequent graphics calls, thus allowing the subpicture to be copied, moved, turned on and off, and tested for light pen hits.

When you want to define a new subpicture by specifying the primitives that compose it, you must issue a SUBP call in the form shown below:

```
CALL SUBP (m1)
```

The graphics calls included after this invocation are used to insert in the display file the lines, points, and images that will make up the subpicture whose tag is m1.

Succeeding graphics calls will apply to subpicture m1 until you issue a call to the ESUB subroutine (see Section 2.4.2) to terminate the subpicture definition. The structure of subpicture definitions thus looks like the following:

```
CALL SUBP (120)
   .
   .
   .
definition
   .
   .
   .
CALL ESUB
   .
   .
   .
CALL SUBP (130)
   .
   .
   .
definition
   .
   .
   .
CALL ESUB
```

Each subpicture defined in a display file must have a unique tag associated with it; the SUBP m1 parameter assigns this tag and must be in range 1 through 32767.

The second form of the SUBP call allows you to define an instance of an existing subpicture by referencing the tag of that subpicture. In the following form:

```
CALL SUBP (m1,m2)
```

m1 assigns the tag of the new subpicture being defined and m2 references the tag of an existing subpicture. Internally, this SUBP call does not actually duplicate the code of subpicture m2 for use by subpicture m1. Instead, it creates for m1 an instruction in the display file that will execute the code for subpicture m2. Subpicture m1 is considered an instance of subpicture m2. Because this form of SUBP does not begin a definition, you should not include a corresponding call to ESUB when using this SUBP format.

A new subpicture may often be constructed by referencing several existing subpictures. For this reason, the DECgraphic-11 package supports a nested subpicture call facility. A subpicture may contain a call to another subpicture, and calls may be nested to a depth of eight. To create a nested subpicture call, begin the definition with a call to SUBP with one argument, followed by additional calls to SUBP, before concluding the definition with an ESUB call. Including the first call to SUBP, there may be as many as eight calls to SUBP before any calls to ESUB.

There must be one call to ESUB for every definition call to SUBP, that is, every call that contains only a single argument. The first call to ESUB encountered in the code will end the last subpicture created, as in a typical subroutine structure.

The following diagram illustrates the nesting of four subpictures in a FORTRAN program.

```
CALL SUBP (201)          begins definition
CALL SUBP (202,721)      references 721
CALL SUBP (203)          begins definition
     .
     .
     .
definition
     .
     .
     .
CALL SUBP (204,771)      references 771
CALL ESUB                terminates 203 definition
CALL ESUB                terminates 201 definition
```

Nested subpictures are useful in applications in which several component graphics are frequently copied or tested for light pen hits, both individually and as a group. The following error message is generated if more than eight subpictures are nested:

```
MORE THAN 8 NESTED SUBP
```

Two examples of the use of SUBP are included at the end of the description of the ESUB subroutine (see Section 2.4.2)

2.4.2 ESUB: Terminating a Subpicture

Form: CALL ESUB [(m)]

The ESUB subroutine is used to terminate a subpicture definition. The relation of this subroutine to SUBP is described fully in Section 2.4.1. The m parameter is only relevant in VS60 systems but provides a useful reference in VT11 systems as well. If m is included, DECgraphic-11 restores the values of any display file status parameters that may have been reset for use in the subpicture that ESUB terminates. The m parameter may be any value and need not have any particular significance in the application program. It is considered good programming practice to set m to the tag of the subpicture being terminated, but this is not a requirement.

Examples:

This example defines a subpicture whose tag is 1000. This subpicture draws an X on the display screen. Note that the call to ESUB terminates the subpicture definition. A call to ESUB with an argument (1000 in the example) causes the subpicture to be terminated and the display status parameters (light pen enable, intensity, flash mode, line type) to be restored to the values in effect before the subpicture was invoked.

```
      .
      .
      .
CALL SUBP (1000)
CALL VECT (100.,100.)
CALL VECT (0.,-100.,,-10)
CALL VECT (-100.,100.)
CALL ESUB (1000)
      .
      .
      .
```

The following example generates two calls to the subpicture whose tag is 1000, and positions the subpicture instances at screen coordinates (0.,0.) and (500.,500.). The new subpictures will have tags 1001 and 1002 respectively. Note that any changes to subpicture 1000 will change the 1001 and 1002 references as well.

```
      .
      .
      .
CALL APNT (0.,0.,,-4)
CALL SUBP (1001,1000)
CALL APNT (500.,500.,,-4)
CALL SUBP (1002,1000)
      .
      .
      .
```

2.4.3 COPY: Copying a Subpicture

Form: COPY ([m1],m2)

The COPY subroutine allows you to create a copy of an existing subpicture and assign a tag to a new subpicture. The m2 parameter represents the tag of the existing subpicture. If m1 is included, it represents the tag assigned to the copied subpicture. If the m1 parameter is omitted from the call, subpicture m2 is copied to the currently open subpicture.

For example, in the following:

```
CALL SUBP (122)
CALL COPY (,701)
CALL ESUB
```

subpicture 122 has been "opened" for definition, so subpicture 701 is copied and assigned the tag 122. An alternative way of expressing this operation is:

```
CALL COPY (122,701)
```

NOTE

The new subpicture will not be affected by subsequent changes to the subpicture that was copied. As in SUBP, coordinate references in the subpicture being copied should be to relative, not absolute, positions on the display screen. Avoid using APNT and AVECT in subpictures to be copied.

Example:

This example positions the beam at (800.,800.), copies subpicture 1000, and assigns tag 1003 to the new subpicture. Any subsequent modifications to subpicture 1000, by means of CHANGE, ERAS, or another routine, will not affect subpicture 1003.

```
    .
    .
    .
CALL APNT (800.,800.,,-4)
CALL COPY (1003,1000)
    .
    .
    .
```

2.4.4 OFF: Turning Off a Subpicture

Form: CALL OFF (m)

The OFF subroutine is used to turn off the subpicture whose tag is m. When a subpicture is turned off, there will be two major effects on the display:

1. The graphics calls included in the subpicture will not produce any output on the screen.

2. If subpicture m is terminated by a nonrestoring ESUB (i.e., one without an m parameter), the display status parameters (e.g., beam position, intensity, line type) will reset to their default values by the call to OFF.

Because of the second condition, any call to OFF should be followed by a call to APNT to set the beam position and change line type and other display parameters, unless the subpicture was ended with a restoring ESUB or unless it is desirable to have these changes occur.

When a subpicture is turned off, the definition of the subpicture remains in the display file, so the subpicture may be copied even though it is turned off. A turned off subpicture may be turned on again by means of the ON subroutine (see Section 2.4.5). Turning off a subpicture that is already turned off has no effect.

To increase system speed, it may be desirable to turn off a subpicture before it has been completely defined, that is, between the SUBP and ESUB calls that define it. This is allowed by DECgraphic-11 and is often useful when building a display that should be seen only when its construction is complete.

An example of using OFF is included in the description of the ON subroutine (see Section 2.4.5).

2.4.5 ON: Turning On a Subpicture

Form: CALL ON (m)

The ON subroutine is used to turn on the subpicture whose tag is m. ON is used to redisplay a subpicture that has been turned off by means of the OFF subroutine. Turning on a subpicture that is already on has no effect.

Example:

This example builds subpicture 1500 from the data in arrays X and Y. The subpicture is turned off until the entire picture has been constructed, and then the display is turned on. This gives you the facility to display "all-at-once" pictures, as well as to use "growing" picture drawing effects.

```
          .
          .
          .
      CALL APNT (0.,500.,,-4)
      CALL SUBP (1500)
      CALL OFF (1500)
      DO 100 I=1,50
100   CALL VECT (X(I),Y(I))
      CALL ESUB (1500)
      CALL ON (1500)
          .
          .
          .
```

2.4.6 ERAS: Erasing a Subpicture

Form: CALL ERAS [(m)]

The ERAS subroutine is used to turn off subpicture m and to erase the definition of the subpicture in the display file. The OFF subroutine turns off the subpicture but does not perform the erase operation.

ERAS eliminates references to tag m in the display file, thus making the tag available for use in other definitions. Erasing a subpicture

does not recapture the portion of the display file used for subpicture definition. However, file space occupied by erased subpictures may be condensed by means of the CMPRS (see Section 2.9.1) or SAVE (see Section 2.9.2) subroutines.

NOTE

Erasing a subpicture also erases all references to it made in calls to SUBP that have two arguments.

If you specify a call to ERAS without an argument, this will have the effect of erasing the tracking object on the display screen.

Example:

This example creates a new subpicture whose tag is 25, copies into it the contents of subpictures 100 and 200, and then erases 100 and 200. This allows the space used by 100 and 200 to be reclaimed by the CMPRS subroutine. Note that subpicture 25 is turned off until subpictures 100 and 200 are erased to prevent the two subpictures from appearing overly bright due to displaying an image multiple times.

```
      .
      .
      .
CALL SUBP (25)
CALL OFF (25)
CALL COPY (,100)
CALL COPY (,200)
CALL ESUB (25)
CALL ERAS (100)
CALL ERAS (200)
CALL ON (25)
      .
      .
      .
```

2.4.7 NMBR: Creating a Numeric Subpicture

Form: CALL NMBR (m,var[,n,format])

The NMBR subroutine creates a special numeric subpicture that can be displayed on the screen in any FORTRAN format and can be updated in "odometer" fashion. The m parameter identifies the tag to be assigned to the subpicture, and var is the FORTRAN variable containing the numeric data to be output on the display screen. The number is displayed with a maximum field width of 16, in any legal format that you specify.

The NMBR call may be issued with two or four parameters. The simpler form of the call:

```
CALL NMBR (m,var)
```

creates a subpicture with tag m and displays the number included in var in the default format. This default is originally set to F16.8, but is reset by any user calls to NMBR; this is explained in greater detail below.

You may provide an explicit format specification in the NMBR call. The n parameter represents the format width, and format is any legal FORTRAN format. The format specification included in the NMBR call is identical to the FORTRAN specification, including the use of enclosing parentheses. The format specification is included as a string, as illustrated below.

```
CALL NMBR (200,X2(I),8,'(F8.2)')
```

If a format specification is not provided in an NMBR call, the number is displayed in the format last specified in an NMBR call in the program. The second NMBR call below will cause the value of B to be output in format F10.4.

```
CALL NMBR (101,A,10,'(F10.4)')
   .
   .
   .
CALL NMBR (102,B)
```

References to tags associated with existing numeric subpictures will update these subpictures.

As with other types of graphics output, you may position the numeric data on the screen by moving the beam to the desired location before issuing a call to NMBR. The lower left corner of the first digit displayed will start at the current beam position.

A further example of using NMBR is included in the description of the CMPRS subroutine (see Section 2.9.1).

2.4.8 CVSCAL: Scaling Subpicture Characters and Vectors

Form: CALL CVSCAL (m[,ifc,ifv])

On the VS60, the CVSCAL subroutine scales the size of characters and vectors displayed in subpicture m to the values specified in ifc and ifv. Calls to CVSCAL have no effect in VT11 systems. This subroutine allows you to enlarge or contract the characters and vectors output on the display screen. This may be helpful in creating charts or figures whose components may be enlarged for more detailed examination or in creating and labeling segments to be contracted for inclusion in a larger structure.

In the CVSCAL call, ifc represents the character scaling factor. The value of ifc must be an integer in range 1 through 4, and characters may be scaled in increments of one-half. The size of characters in subpicture m may vary from one-half normal size to twice normal size, as shown in the list below. The standard size is indicated by an ifc value of 2.

ifc Value	Character Size
1	1/2 normal size
2	normal size
3	1 1/2 normal size
4	2 times normal size

The ifv argument represents the vector scaling factor. The value of ifv must be an integer in range 1 through 15, and vectors may be scaled in increments of one-quarter. The size of line segments in subpicture m may vary from one-fourth normal size to three and three-fourths normal size, as shown below. The standard size is indicated by an ifv value of 4.

ifv Value	Vector Size
1	1/4 normal size
2	1/2 normal size
3	3/4 normal size
4	normal size
5	1 1/4 normal size
6	1 1/2 normal size
7	1 3/4 normal size
8	2 times normal size
9	2 1/4 normal size
10	2 1/2 normal size
11	2 3/4 normal size
12	3 times normal size
13	3 1/4 normal size
14	3 1/2 normal size
15	3 3/4 normal size

In addition to scaling the characters or vectors in the specified subpicture, CVSCAL will scale the rest of the display file as well, unless you terminate the subpicture definition with an ESUB call that contains an m parameter.

Examples:

This example draws a box in the center of the screen and changes its size each time the operator enters a carriage return, until the size reaches the maximum (i.e., vector scale 15).

```
        .
        .
        .
      CALL APNT (412.,412.,,-4)
      CALL SUBP (100)
      CALL LVECT (200.,0.)
      CALL LVECT (0.,200.)
      CALL LVECT (-200.,0.)
      CALL LVECT (0.,-200.)
      CALL ESUB (100)
      DO 200 I=1,15
      PAUSE
200   CALL CVSCAL (100,,I)
        .
        .
        .
```

The next example displays the string 'COMMANDS' in characters twice the normal size. It then displays 'SAMPLE', 'ANALYZE', 'PLOT', and 'STOP' in a column beneath 'COMMANDS' in characters of normal size.

```
        .
        .
        .
     CALL APNT (0.,800.,,-4)
     CALL SUBP (10)
     CALL TEXT ('COMMANDS:',1)
     CALL ESUB (10)
     CALL CVSCAL (10,4)
     CALL TEXT (4,'SAMPLE',1,'ANALYZE',1,
    X  'PLOT',1,'STOP')
        .
        .
        .
```

2.5 DISPLAYING GRAPHS AND FIGURES

The subroutines described in this section allow you to specify simple calls that generate more complex structures on the display screen than the primitive elements described in Section 2.3. These routines generate subpictures that are used to display special kinds of graphs and figures. They allow you to display the contents of an entire array with one simple subroutine call. Because graphs and figures are subpictures, you can erase them and turn them on and off. You can also modify a display by changing the values of certain primitives defined in subpictures.

2.5.1 XGRA: Displaying an X-Axis Graph

Form: CALL XGRA (dy,A,n,m[,l,i,f,t])

The XGRA subroutine creates a special graph subpicture with tag m, which consists of a series of points. XGRA allows you to display a graph whose y-coordinates are evenly spaced on the display screen. The coordinates to be plotted on the x-axis are supplied in the first n elements of the array indicated by A in the argument list included above. A must be a one-dimensional real array. The y-coordinates of the graph displayed by XGRA are generated from integral multiples of dy. The x coordinates of the graph being displayed are determined by the x value of the current beam position; the first y coordinate is at the current y value plus the value of dy.

The subpicture created by XGRA is displayed on the screen or can be referenced by subsequent calls to the AGET, APUT (see Sections 2.5.4 and 2.5.5), GET and CHANGE (see Sections 2.6.3 and 2.6.4) subroutines. These routines are used to access the array of graph data in order to change the primitive elements stored there.

When a graph subpicture is created by means of XGRA, the x-array data is entered in the display file in absolute coordinate positions, thus making the array reusable.

In the call to XGRA, you may optionally specify new values for the display file parameters l,i,f, and t, described in Section 2.3.

2.5.2 YGRA: Displaying a Y-Axis Graph

Form: CALL YGRA (dx,A,n,m[,l,i.f,t])

The YGRA subroutine is used to create a special graph subpicture with tag m, which consists of a series of points. YGRA allows you to display a graph whose x-coordinates are evenly spaced on the display screen. The coordinates to be plotted on the y-axis are supplied in the first n elements of the array indicated by A in the format included above. A must be a one-dimensional real array. The x-coordinates of the graph displayed by YGRA are generated from integral multiples of dx. The y coordinates of the graph being displayed are determined by the y value of the current beam position; the first x coordinate is at the current x value plus the value of dx.

Like XGRA, YGRA creates a subpicture whose primitive elements can be accessed and changed by means of the AGET, APUT, GET, and CHANGE subroutines.

When a subpicture graph is created by means of YGRA, the y-array data is entered in the display file in absolute coordinate positions, thus making the array reusable.

In the call to YGRA, you may optionally specify new values for the display file parameters l,i,f, and t.

Example:

Draw a sine wave across the entire screen using 51 points.

```
      COMMON/DFILE/IBUF(200)
      DIMENSION R(51)
      DATA PI/3.1415926535/
      DO 1 I=1,51
1      R(I)=200.+200.*SIN(PI*(I-1)/25.)
      CALL INIT(200)
      CALL YGRA(20.,R,51,1000)
      STOP
      END
```

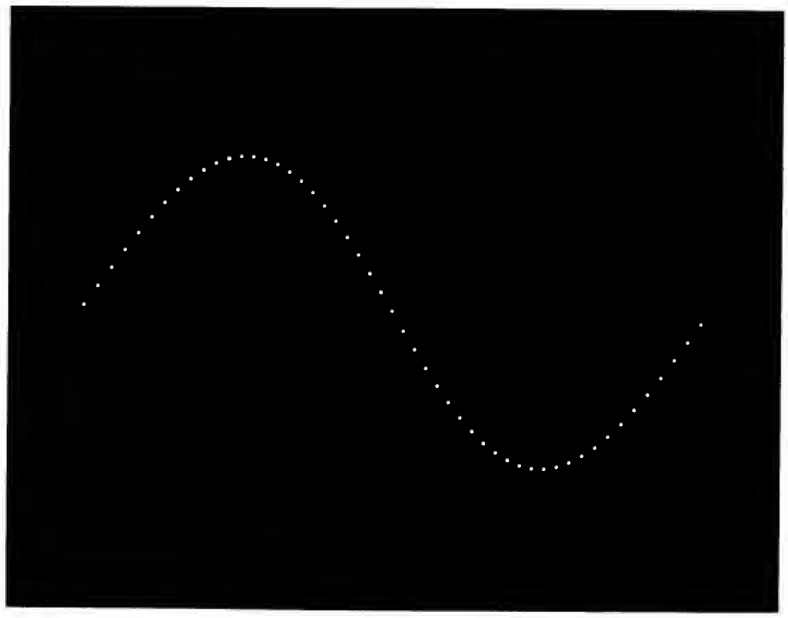

Figure 2-3 YGRA Subroutine

2.5.3 FIGR: Displaying a Figure

Form: CALL FIGR (A,n,m[,l,i,f,t])

The FIGR subroutine allows you to create a special figure subpicture with tag m from an array of (x,y) increment pairs. The figure is plotted from the first n elements of the array indicated by A. A must be a one-dimensional real array consisting of pairs of relative (x,y) values. (A(l),A(2)) is the first (x,y) pair to be plotted, (A(3),A(4)) is the second (x,y) pair, and so on. The first line segment in the figure is drawn from the current beam position to (A(l) A(2)).

FIGR creates a subpicture which is displayed on the screen and can be referenced by subsequent calls to the AGET, APUT, GET, and CHANGE subroutines. These routines are used to access the array of vector data in order to change the primitive elements stored there.

In the call to FIGR, you may optionally specify new values for the display file parameters l,i,f, and t.

Example:

The following example uses an "invisible" figure to move a graphics display.

```
          COMMON/DFILE/IBUF(200)
          DIMENSION R(2)
          .
          .
          .
          CALL FIGR(R,2,1001,0,-5)
          .
          .
          .
     C    MOVE THE GRAPHICS
          WRITE (5,1)
     1    FORMAT (' ENTER NEW COORDINATES')
          READ (5,2) X,Y
     2    FORMAT (2F7.2)
          CALL APUT(1001,1,X)
          CALL APUT(1001,2,Y)
          .
          .
          .
```

2.5.4 AGET: Returning the Value of a Primitive

Form: CALL AGET (m,j,Z)

The AGET subroutine is used to access an array containing graph or figure data in order to examine the value of a primitive in subpicture m.

You may index into the array represented by A in an XGRA, YGRA, or FIGR subroutine call by specifying an array subscript j in the AGET call. AGET accesses the jth element of the array from which subpicture m was created and returns the value of the element in variable Z.

2.5.5 APUT: Changing the Value of a Primitive

Form: CALL APUT (m,j,b)

The APUT subroutine accesses an array containing graph or figure data in order to change a primitive in subpicture m. APUT allows you to alter graphics output while it is being displayed by XGRA, YGRA, or FIGR.

You may index into the array represented by A in an XGRA, YGRA, or FIGR subroutine call by specifying an array subscript j in the APUT call. APUT accesses the jth element of the array from which subpicture m was created and changes that element to the value

supplied in parameter b. The display of subpicture m is updated immediately to reflect the change in the primitive definition.

2.5.6 FPUT: Changing and Adjusting the Value of a Primitive

Form: CALL FPUT (m,j,b)

The FPUT subroutine is used to access an array containing figure data in order to change a primitive in subpicture m. FPUT allows you to alter graphics output while it is being displayed by FIGR. It also adjusts the next element of the subpicture to ensure that subsequent points in the subpicture will be at the same absolute coordinate positions on the screen.

You may index into the array represented by A in the FIGR subroutine call that created the figure by specifying an array subscript j in the FPUT call. FPUT accesses the jth primitive of the array from which subpicture m was created and changes that primitive to the value supplied in parameter b. The display of subpicture m is updated immediately to reflect the change in the primitive definition.

2.6 USING DISPLAY FILE POINTERS

The subroutines described in this section allow you to manipulate pointers to elements in the display file and to change the values of the primitives referenced by those pointers. The first three subroutines in this category, POINTR, ADVANC, and GET, do not actually display output on the screen. They simply allow you to establish, advance, and return information on pointers to primitives in the display file. This facilitates the alteration of the values of vector, point, or character primitives by the other graphics calls described in this section.

There are 21 distinct pointers in the DECgraphic-11 system. You can manipulate all but the 21st pointer, which serves as the system pointer and should not be referenced in a user program.

2.6.1 POINTR: Setting Up a Pointer

Form: CALL POINTR (k,m[,j])

The POINTR subroutine sets up pointer number k at the jth primitive of the subpicture whose tag is m. The value of the k parameter identifies the particular pointer being established and must be in range 1 through 20. If the j parameter is omitted from the POINTR call, pointer k will be set to the first primitive of subpicture m in the display file. If you specify a value for j greater than the number of primitives in the subpicture, subsequent primitives will be inserted at the end of the subpicture definition.

NOTE

Pointers never reference subpictures, subpicture calls, or erased primitives. When these elements are encountered in the display file, the pointers skip them automatically.

One example of using POINTR is included below. Others are in the descriptions of GET (see Section 2.6.3) and GRID (see Section 2.8.6).

Example:

This example draws a vector that continuously sweeps out a circle in the center of the screen. The vector moves in a counter-clockwise direction.

```
          .
          .
          .
       CALL APNT (512.,512.,,-4)
       CALL SUBP (100)
    -- CALL VECT (100.,0.)
       CALL ESUB
          .
          .
          .
       CALL POINTR (3,100)
       ANGLE=0.
10     CALL CHANGE (3,100.*COS(ANGLE),100.*SIN(ANGLE))
       ANGLE=AMOD (ANGLE+.01,6.28)
       GO TO 10
          .
          .
          .
```

2.6.2 ADVANC: Advancing a Pointer

Form: CALL ADVANC (k[,n])

The ADVANC subroutine allows you to advance the position of a pointer in the display file. The k parameter identifies the particular pointer to be advanced and must be in range 1 through 20. If the n parameter is included, the pointer is advanced by n primitives from its current position. If n is omitted, the pointer advances by one.

A pointer cannot be advanced beyond the end of a subpicture definiton in the display file. If an ADVANC call attempts to move a pointer beyond the subpicture boundary, subsequent primitives will be inserted at the end of the subpicture definition. ADVANC will skip any nested subpicture definitions in the subpicture being examined.

An example of using ADVANC is included in the description of the GET subroutine (see Section 2.6.3).

2.6.3 GET: Returning the Coordinates of a Primitive

Form: CALL GET (k,X,Y)

The GET subroutine is used to return the coordinates of the primitive currently being referenced by a pointer. The k parameter identifies the particular pointer whose primitive coordinates are to be returned and must be in range 1 through 20. The (x,y) coordinates are returned in variables (X,Y).

Example:

This example draws a sine wave of dots and then shrinks it one point at a time, using pointer 4 to move through the subpicture definition.

```
          .
          .
          .
      CALL SUBP (1000)
      X=0
      DO 10 I=1,51
      CALL APNT (X,SIN(.125*(I-1))*500.+500.)
10    X=X+20.
      CALL ESUB
      CALL POINTR (4,1000)
      DO 20 I=1,51
      CALL GET (4,X,Y)
      CALL CHANGE (4,X,Y/2.)
20    CALL ADVANC (4)
          .
          .
          .
```

2.6.4 CHANGE: Changing the Coordinates of a Primitive

Form: CALL CHANGE (k,x,y)

The CHANGE subroutine is used to change the coordinates of the primitive referenced by pointer k. The coordinates are changed to the new value specified in coordinate positions (x,y). The value of k must be in range 1 through 20.

This subroutine is usually called to change a primitive defined by the VECT, SVECT, LVECT, RPNT, APNT, or WINDOW subroutines or a graph subpicture created by XGRA or YGRA. When a CHANGE call is specified for a graph created by XGRA or YGRA, the x value is ignored in the XGRA call, and the y value is ignored in the YGRA call.

2.6.5 CHANGA: Changing a Primitive and Adjusting the Next Primitive

Form: CALL CHANGA (k,x,y)

The CHANGA subroutine allows you to change the coordinates of the primitive referenced by pointer k to the new value specified in coordinate positions (x,y). Unlike the CHANGE subroutine (see Section 2.6.4), it also adjusts the next primitive in the display file so that subsequent images will be at the same absolute screen positions. The value of k must be in range 1 through 20.

This subroutine may be called to change a primitive defined by the VECT, SVECT, LVECT, RPNT, APNT, or WINDOW subroutines or a graph subpicture created by XGRA or YGRA. When a CHANGA call is specified for a graph created by XGRA or YGRA, the x value is ignored in the XGRA call, and the y value is ignored in the YGRA call.

Example:

This example reads a primitive number and a new value from the keyboard. It then changes the y coordinate of a single primitive of subpicture 100 to the new value and adjusts the following primitive.

```
        .
        .
        .
1000  WRITE (5,1010)
1010  FORMAT (' INPUT PRIMITIVE NUMBER AND NEW VALUE')
      READ (5,1020)I,V
1020  FORMAT (I5,F10.5)
      CALL POINTR (1,100,I)
      CALL GET (1,X,Y)
      CALL CHANGA (1,X,V)
      GO TO 1000
        .
        .
        .
```

2.6.6 CHANGT: Changing the Value of a Character Primitive

Form: CALL CHANGT (k,a1[,a2...])

The CHANGT subroutine is used to change the value of the character primitive referenced by pointer k. The primitive changed by CHANGT is usually one created by the TEXT subroutine (see Section 2.3.7). The value of k must be in range 1 through 20.

The values of parameters a1,a2, and so on are constructed exactly as the a parameters in the TEXT subroutine are. Each value may be a character string, a carriage return/line feed count, or a control code indicating the use of shift-out characters, italic or rotation mode, or superscripted or subscripted characters.

Example:

This example reads a word from the keyboard and positions it in the subpicture.

```
        .
        .
        .
     LOGICAL *1 WORD (80)
     DATA WORD/79*'.',0/
        .
        .
        .
     CALL SUBP (100)
     CALL APNT (0.,750.,,-4)
     CALL TEXT ('THE WORD IS:',1)
     CALL TEXT (WORD)
     CALL ESUB (100)
        .
        .
        .
     CALL POINTR (10,100,3)
     READ (5,10)N,(WORD(I),I=1,N)
 10  FORMAT (I2,80A1)
     WORD (N+1)=0
     CALL CHANGT (10,WORD)
        .
        .
        .
```

2.6.7 INSERT: Inserting Graphic Elements in the Display File

Form: CALL INSERT [(k)]

The INSERT subroutine allows you to insert graphic elements in the existing display file just before the position of the primitive referenced by pointer k. This enables you to add new material to the file, not merely to replace existing primitive elements.

If the k parameter is included an INSERT call effectively reopens the display file for input and causes subsequent graphics calls to define the elements to be inserted in the file. If the k parameter is omitted, the insert operation is terminated, the display file is effectively closed, and any subsequently defined graphics elements will be inserted at the end of the file in the ordinary way.

NOTE

After you have issued an INSERT call and before you have moved the pointer, you should not issue calls to the CMPRS, SAVE, RSTR, SUBP, SENSE, INTENS, FLASH, LINTYP, CVSCAL, or ESUB subroutines.

An example of using INSERT is included in the description of the ERASP subroutine (see Section 2.6.8).

2.6.8 ERASP: Erasing a Primitive

Form: CALL ERASP (k)

The ERASP subroutine is used to erase the display file primitive referenced by pointer k. After erasing the referenced primitive, ERASP positions k at the next primitive in the file.

NOTE

Because primitive numbers are relative, erasing a primitive changes the number of subsequent primitives in the subpicture. The number is decreased by one.

Example:

This example waits for a light pen hit on a subpicture and then erases the primitive referenced by the hit. A vector is inserted at this point and attached to the tracking object. The program then waits for a carriage return from the keyboard to adjust the vector.

```
        .
        .
        .
100   CALL LPEN (IH,IT,X,Y,IP)
      IF (IH.EQ.0) GO TO 100
      CALL POINTR (2,IT,IP)
      CALL ERASP (2)
```

```
      CALL INSERT (2)
      CALL LVECT (0.,0.)
      CALL TRAK (X,Y)
      CALL POINTR (2,IT,IP)
      CALL ATTACH (2)
      CALL INSERT
      WRITE (5,101)
101   FORMAT (' REPOSITION TRACKING OBJECT, TYPE <CR> WHEN DONE')
      READ (5,110) I
110   FORMAT (A1)
         .
         .
         .
```

2.7 ALTERING DISPLAY FILE STATUS PARAMETERS

The subroutines described in this section allow you to alter the following basic display file status parameters for use by specific primitives:

- light pen enable
- intensity level
- flash mode
- line type

These parameters may also be changed by a variety of other graphics calls, especially those described in Section 2.3 and 2.5. A detailed description of the meaning of the l,i,f, and t parameters is included in Section 2.3 and is not repeated below.

2.7.1 SENSE: Setting the Light Pen Parameter

Form: CALL SENSE (k[,l,s])

The SENSE subroutine is used to enable or disable light pen interaction for the primitive referenced by pointer k. The k parameter must be in range 1 through 20.

If the l parameter included in the SENSE call is positive, or if l is omitted from the call, light pen interaction is enabled for the referenced primitive. A light pen interrupt will occur when the light pen is pointed at a primitive on the display screen that has been made light pen-sensitive. If the value of the l parameter is negative, light pen interaction is disabled for the primitive referenced by k. If l is zero, the current light pen status remains unchanged.

The s parameter is included in VS60 systems to indicate the scope to which the light pen reference applies; legal values are 1 and 2. If the s parameter is omitted, scope 1 is assumed.

An example of using SENSE is included in the description of INTENS (see Section 2.7.2).

2.7.2 INTENS: Setting the Intensity Parameter

Form: CALL INTENS (k[,i])

The INTENS subroutine allows you to specify the intensity level of the primitive referenced by pointer k. The k parameter must be in range 1 through 20.

If the i parameter included in the INTENS call is in range 1 through 8, the referenced primitive is intensified and the intensity level is changed to the value of i; 1 is the faintest intensity and 8 is the brightest. If i is zero, omitted from the call, or greater than 8, the referenced primitive is intensified but the intensity level is not changed. If i is in range -1 through -8, the referenced primitive is not intensified but the intensity level is changed to the absolute value of i. If i is less than -8, the referenced primitive is not intensified and the intensity level is not changed.

Example:

This example enables light pen sensitivity for all subpictures (1 through NOBJ). It then waits for a light pen hit, with the tip switch depressed, on one of the subpictures. The light pen enable is then turned off, and the intensity level of the subpicture selected by the light pen is increased.

```
          .
          .
          .
      DO 100 I=1,NOBJ
      CALL POINTR (7,I)
100   CALL SENSE (7,1)
150   CALL LPEN (IH,IT,,,,,,ITIP)
      IF (IH.EQ.0.OR.IT.GT.NOBJ.OR.ITIP.EQ.0) GO TO 150
      DO 200 I=1,NOBJ
      CALL POINTR (7,I)
200   CALL SENSE (7,-1)
      CALL POINTR (7,IT)
      CALL INTENS (7,8)
          .
          .
          .
```

2.7.3 FLASH: Setting the Flash-Mode Parameter

Form: CALL FLASH (k[,f])

The FLASH subroutine is used to enable or disable flash or blink mode for the primitive referenced by pointer k. The k parameter must be in range 1 through 20.

If the f parameter included in the FLASH call is positive, or if f is omitted from the call, flash mode is enabled. The display for the referenced primitive and for subsequent graphics elements will blink when output on the screen.

If the value of the f parameter is negative, flash mode is disabled for the primitive referenced by k and for subsequent graphics elements. If f is zero, the current flash mode status remains unchanged.

An example of using FLASH is included in the description of LPEN (see Section 2.8.1).

2.7.4 LINTYP: Setting the Line-Type Parameter

Form: CALL LINTYP (k[,t])

The LINTYP subroutine allows you to specify the line type of the primitive referenced by pointer k. The k parameter must be in range 1 through 20.

If the t parameter included in the LINTYP call is in range 1 through 4, the line type is changed for the referenced primitive and for subsequent graphics elements. Any vector will be drawn according to the following conventions:

t Value	Vector Type
1	Solid lines
2	Long-dashed lines
3	Short-dashed lines
4	Dot-dash lines

If t is zero, omitted from the call, negative, or greater than 4, the current line type remains unchanged.

2.8 FACILITATING LIGHT PEN INTERACTION

The subroutines in this category allow you to use the interactive light pen facilities available in the DECgraphic-11 system. These routines are used to test for light pen hits on the display screen and to position and manipulate a tracking object on the screen.

2.8.1 LPEN: Recording a Light Pen Hit

Form: CALL LPEN (IH,IT[,X,Y,IP,IA,IM,IT1,IT2])

The LPEN subroutine is used to test for a light pen hit on the display screen. It is possible to test for a hit if any of the graphic elements displayed on the screen has been made light pen-sensitive (e.g., by means of the SENSE subroutine; see Section 2.7.1).

If a successful light pen hit is recorded, LPEN returns information on the position and status of the hit in the user variables specified in the subroutine call. These variables are described below.

Variable	Meaning
IH	Flag used to indicate the occurrence of a light pen hit. IH=0: no light pen hit. IH=1: light pen hit on scope 1. IH=2: light pen hit on scope 2.
IT	Tag of the subpicture in which the light pen hit occurred (1-32767). IT=-2: no subpicture.

Variable	Meaning
X,Y	Coordinates of the light pen hit on the display screen (in user coordinate system).
IP	Number of the primitive within the subpicture at which the light pen hit occurred.
IA	Array which contains the precedents of the subpicture in which the light pen hit occurred. The most recent precedent is returned in the first array element (IA(1)). The maximum number of precedents returned by LPEN is seven, plus a terminator to indicate the last entry in the array. The terminator always has the value -2. IA(1)=-2: no subpicture precedents, so only the terminator is returned.
IM	Screen area in which the light pen hit occurred. IM=1: light pen hit in main area. IM=2: light pen hit in menu area.
IT1,IT2	Status of the tip switches for scopes 1 (IT1) and 2 (IT2). There is no tip switch on the VT11 light pen, so the tip switch (IT1) is always on for this device. ITn=0: tip switch off. ITn=1: tip switch on.

Example:

This example displays a menu of program options. It then waits for a light pen hit in the menu area, flashes the option selected by the hit, and branches to the appropriate section of the program.

```
      .
      .
      .
      CALL MENU (,800.,-50.,100,'SAMPLE','FFT','PLOT','STOP')
      .
      .
      .
100   CALL LPEN (IH,IT)
      IF (IH.EQ.0.OR.IT.LT.100.OR.IT.GT.103) GO TO 100
      CALL POINTR (11,IT)
      CALL FLASH (11)
      GO TO (100,200,300,400), IT-99
      .
      .
      .
```

2.8.2 **TRAK:** Placing a Tracking Object on the Screen

Form: CALL TRAK (x,y[,s])

The TRAK subroutine allows you to position a diamond-shaped tracking object on the display screen. The tracking object may not be positioned in the menu area of the VS60 screen. The tracking object is initially positioned on the coordinate position represented by (x,y). For VS60 systems, the s parameter identifies the scope (1 or 2) on which the object is displayed; if s is omitted from the TRAK

call, the default is 1. Once you have placed the tracking object on the screen, it will center itself on any light pen hit within the boundaries of its shape.

NOTE

On VS60 systems, tracking is only effective if you have turned on the light pen tip switch by pressing it onto the surface of the display screen.

You may remove the tracking object from the screen by specifying an ERAS call without an argument (see Section 2.4.6).

An example of using TRAK is included in the description of GRID (see Section 2.8.6).

2.8.3 TRAKXY: Returning the Position of the Tracking Object

Form: CALL TRAKXY (X,Y[,s])

The TRAKXY subroutine returns the current coordinate position of the tracking object on the display screen. The position is returned in variables (X,Y). If you have a VS60 system, you may specify in parameter s the scope (1 or 2) on which the desired tracking object is positioned; if s is omitted from the TRAKXY call, the default is 1. An example of using TRAKXY is included in the description of GRID (see Section 2.8.6).

2.8.4 ATTACH: Attaching a Primitive to the Tracking Object

Form: CALL ATTACH (k[,n,s])

The ATTACH subroutine allows you to attach to the tracking object the primitive graphics element referenced by pointer k. If you have a VS60 system, you may specify in parameter s the scope (1 or 2) on which the tracking object is positioned; if s is omitted from the ATTACH call, the default is 1.

Primitive elements that may be attached in this fashion include the long vector, the absolute point, and the absolute vector. If you specify a pointer to any other primitive in the display file, the ATTACH call will be ignored. By attaching a primitive element to the tracking object, you can easily change the coordinates of a point or vector by moving it to another part of the display screen and then detaching it.

If the primitive to be attached is a long vector, it will be attached to the tracking object and will follow it as the object moves on the display screen. The optional n parameter determines which end of the vector will move and which end will be stationary. If n is positive or omitted from the call, the vector's origin is stationary and its destination will move. If the value of n is negative, the vector's destination is stationary and its origin will move. If you attach a vector that is longer than 1023 raster units, the vector will not reliably move along with the tracking object.

The n parameter need not be specified if the referenced primitive is an absolute point or an absolute vector. If an absolute point or vector is attached to the tracking object, the point or vector end point will be moved the same distance as the tracking object.

You may attach as many as 40 primitives to the tracking object on the display screen by simply issuing repeated calls to ATTACH.

An example of using ATTACH is included in the description of GRID (see Section 2.8.6).

2.8.5 DETACH: Detaching Primitives from the Tracking Object

Form: CALL DETACH [(s)]

The DETACH subroutine is used to detach from the tracking object all primitive elements that are currently attached to the object. If you have a VS60 system, you may specify in parameter s the scope (1 or 2) on which the tracking object is positioned; if s is omitted from the DETACH call, the default is 1.

An example of using DETACH is included in the description of the GRID subroutine (see Section 2.8.6).

2.8.6 GRID: Positioning the Tracking Object on the Grid

Form: CALL GRID (gx,gy[,s])

The GRID subroutine defines the coordinates of a grid of evenly spaced imaginary dots on the display screen and moves the tracking object to the nearest point on the grid. This routine is often used when you are drawing with the light pen or positioning objects on the screen. It also allows you to detect light pen hits that are in the approximate vicinity of one of the points on your grid.

You may specify the x and y spacing of the grid by including the gx and gy parameters in the call to GRID. If you have a VS60 system, you may specify in parameter s the scope (1 or 2) on which the tracking object is positioned; if s is omitted from the GRID call, the default is 1.

When the GRID subroutine moves the tracking object on the screen, it also makes necessary adjustments to the coordinate positions of any points and vectors that are attached to the object. An automatic DETACH is performed after the adjustment.

Example:

This example uses input from the light pen to build a subpicture consisting of an absolute point and a long vector. The start of the line is initially positioned by moving the tracking object and typing a carriage return on the terminal when finished. The GRID subroutine is used to force the start point to lie on the logical grid whose (x,y) spacing is (50.,50.). A long vector is then drawn and attached to the tracking object, which can be moved with the light pen. The vector is then effectively "stretched" with the tracking object to the desired end point. GRID is once again called to normalize the end point.

```
      .
      .
      .
      CALL TRAK (500.,500.)
      WRITE (5,10)
10    FORMAT (' POSITION TRACKING OBJECT, TYPE <CR> WHEN DONE')
      READ (5,20) I
20    FORMAT (A2)
      CALL GRID (50.,50.)
      CALL TRAKXY (X0,Y0)
      CALL SUBP (1500)
      CALL APNT (X0,Y0,,-4)
      CALL LVECT (0.,0.)
      CALL ESUB (1500)
      CALL POINTR (20,1500,2)
      CALL ATTACH (20)
      WRITE (5,30)
30    FORMAT (' DRAW LINE, TYPE <CR> WHEN DONE')
      READ (5,20) I
      CALL GRID (50.,50.)
      .
      .
      .
```

2.9 PERFORMING DISPLAY FILE UTILITY FUNCTIONS

The subroutines in this category are used to perform the following utility functions:

- compressing the display file
- saving the display file
- restoring the display file

2.9.1 CMPRS: Compressing the Display File

Form: CALL CMPRS

The CMPRS subroutine is used to compress space in the display file that is no longer actively required. This is the process sometimes referred to as "garbage collection".

After a portion of a graphics program is executed, the display file may contain a number of primitives and subpictures that are no longer needed and have been erased. The ERAS subroutine does not actually delete elements in the display file, so the space occupied by these erased display elements is not yet available for other purposes. Although the tags associated with erased primitives and subpictures are immediately available, the space in the display file is not. You must periodically issue a CMPRS call to condense the display file and reclaim the space used by the erased display elements.

NOTE

A call to CMPRS invalidates all open pointers and detaches any primitives that are attached to the tracking object.

Example:

This example illustrates a reasonable technique for monitoring the amount of space remaining in the display file; in this case, the amount is expressed as a percentage of the total. It also illustrates the issuing of an automatic CMPRS when this total drops below a certain threshold--10% in the example. The DPTR subroutine is used to determine the amount of space used, and NMBR to display the percentage remaining in the display file.

```
          .
          .
          .
      CALL INIT (2000)
          .
          .
          .
100   CALL DPTR (NEXT)
      PLEFT = (2001.-FLOAT(NEXT))/20.
      CALL APNT (0.,0.,,-4)
      CALL NMBR (2000,PLEFT,5,'(F5.1)')
      IF (PLEFT.GT.10.) GO TO 200
      CALL CMPRS
      CALL DPTR (I)
      IF (I.LT.NEXT) GO TO 200
      WRITE (5,110)
110   FORMAT (' WARNING: DISPLAY FILE NEARLY FULL')
200   CONTINUE
          .
          .
          .
```

2.9.2 SAVE: Saving the Display File

The SAVE subroutine allows you to compress the display file and save it as a data file on a mass-storage device such as disk or floppy disk. The saved file can subsequently be restored as the current display file by means of the RSTR subroutine (see Section 2.9.3).

SAVE is particularly useful in creating a library of files that can be called in from secondary storage as needed for application program purposes. It allows one program to create displays that other programs can access without having to incur the overhead required to create these displays. For example, a display file picture library consisting of a large number of subpictures may be created and saved on disk. When the saved display file is restored, subpictures may be turned on and copied as desired by means of the restoring program's subpicture calls.

The saved display file is named according to the standard naming conventions for RT-11 and RSX-11M. The subroutine call therefore has two possible formats:

For RT-11:

```
CALL SAVE ('[dev:]filename[.ext]')
```

For RSX-11M:

```
CALL SAVE ('[dev:][[g,m]]filename[.ext][;ver]')
```

The format parts in brackets are optional. The size limitations are as follows:

dev:	maximimum of three characters followed by a colon; if dev: is omitted, the devices specified by default are DK: for RT-11 and SY: for RSX-11M.
file name	maximum of six characters in RT-11, the first character a letter; RSX-11M permits up to nine characters, but only the first six are recognized as a unique name.
.ext	maximum of three characters in either system; the defaults are .DAT for RT-11, no extension for RSX-11M.
;ver	version number (RSX-11M only); if omitted, the saved file will have the highest version number for that file name by default.
[g,m]	allows the user (RSX-11M only) to set the user identification code (UIC) for the file; allowable values for group (g) and member (m) are octal integers between 1 and 377; if omitted, the UIC is set to the UIC of the terminal; note that the brackets are part of the UIC and must be included when the UIC is given explicitly.

NOTE

A call to SAVE invalidates all open pointers and detaches any primitives that are attached to the tracking object.

An example of using SAVE is included in the description of the RSTR subroutine (see Section 2.9.3).

2.9.3 RSTR: Restoring the Display File

Forms:

```
RT-11:  CALL RSTR('[dev:]filename[.ext]')
RSX-11M:  CALL RSTR('[dev:][g,m]]filename[.ext][;ver]')
```

The RSTR subroutine is used to restore a display file that has been saved on a mass-storage device by means of the SAVE subroutine (see Section 2.9.2). RSTR copies the saved file into either an empty display file in memory (i.e., one that has been initialized by means of INIT) or a non-empty display file. In the latter case, the restored file is appended to the end of the current display file in memory.

When the restored file is appended to an existing display file in memory, you should be aware that tags in the original file area may duplicate those in the restored file area. In the DECgraphic-11 package, any reference to a tag will always apply to the first occurrence of the tag in the file. To avoid encountering this situation, you should number subpictures extremely carefully.

If you find that two subpictures in display file have the same tag, you can copy the original subpicture with a new tag assignment and then erase the subpicture from which you copied. You will now be able to access the restored subpicture by referencing the old tag and to access the copy of the old subpicture by referencing the new tag.

When the file is restored, it is treated as if the code required to create the file had been entered at this point. After the call to the RSTR subroutine, the basic display file parameters (l,i,f, and t) have the values they were assigned in the program that created the restored file.

NOTE

A call to RSTR invalidates all open pointers and detaches any primitives that are attached to the tracking object.

The filename included in the RSTR call must follow the standard naming conventions described in Section 2.9.2. As in the case of SAVE, the default device is DK: for RT-11 and SY: for RSX-11M.

Example:

This example illustrates the way in which program 1 can create a display and save the display file, and program 2 can restore it.

```
                .
                .
                .
PROGRAM 1       CALL VECT (X,Y)
                .
                .
                CALL STOP
                CALL SAVE ('PICTUR.DSP')
                CALL CONT


PROGRAM 2       COMMON/DFILE/IBUF(2000)
                .
                .
                .
                CALL INIT (2000)
                CALL STOP
                CALL RSTR ('PICTUR.DSP')
                CALL CONT
                .
                .
```

2.10 PERFORMING ADVANCED DISPLAY FILE FUNCTIONS

This section describes three subroutines that have been implemented to aid advanced users of the DECgraphic-11 system. They allow you to access and change any word in the display file by means of array subscripting techniques. These facilities provide very detailed control over the contents of the display file and should be considered advanced graphics features. You should not use them unless you are confident that the operations you are performing will not damage a segment of the display file.

2.10.1 DPTR: Returning the Next Available Display File Position

Form: CALL DPTR (I)

The DPTR subroutine returns the position of the next available word in the display file. It also allows you to determine how much of the display file is currently in use.

DPTR returns an integer value in variable I. This value represents the number of the display file array element in which the next graphic element entered in the display file will be stored. This is the next available word in the file.

An example of using DPTR is included in the description of the CMPRS subroutine (see Section 2.9.1).

2.10.2 DPYNOP: Inserting No-ops in the Display File

Form: CALL DPYNOP (n)

The DPYNOP subroutine allows you to insert any number of no-op instructions in the display file. This sets up the display file for subsequent modification by the DPYWD routine (see Section 2.10.3), which can subscript into the display file array and change specified words.

The n parameter represents the number of no-op instructions to be inserted in the file. The no-ops are inserted beginning at the next available word in the display file. As described in Section 2.10.1, you can determine which word this is by issuing a call to the DPTR subroutine.

2.10.3 DPYWD: Inserting a Data Word in the Display File

Form: CALL DPYWD (i,j)

The DPYWD subroutine is used to insert the 16-bit data word represented by i at the next available word in the display file. When DECgraphic-11 encounters a DPYWD call in which the j parameter has a value of zero, it displays the inserted elements on the screen. If j is nonzero, the insertion is not immediately displayed, thus facilitating the successive insertion of multiple words of data.

CHAPTER 3

PROGRAMMING TECHNIQUES

This chapter summarizes a variety of hints for making your graphics programs more efficient in terms of memory use and execution time. It is divided into two major categories:

- subpicture techniques
- general graphics techniques

Some of the methods described in these categories are also included in Chapter 2, in the discussion of specific FORTRAN subroutines. They are repeated here for ease of reference. These techniques are illustrated in the FORTRAN programing example included in Appendix D.

3.1 SUBPICTURE TECHNIQUES

The programming techniques described in this section should help you in defining and using subpictures in the DECgraphic-11 environment. See the discussion of subpictures in Sections 1.3.4 and 2.4.

3.1.1 Using Subpictures as Subroutines

The DECgraphic-11 package is heavily oriented to the use of subpictures. Approximately half of the FORTRAN graphics subroutines described in Chapter 2 are intended for use in defining and accessing subpictures and primitives within subpictures. For example, when light pen hits are recorded, the position of the hit in a subpicture is returned. Pointers are positioned within subpicture definitions. Characters and vectors may be scaled for use in particular subpictures. Subpictures can be turned on and off and can be erased.

In the DECgraphic-11 environment, subpictures are very similar to subroutines and should be used for the same reasons that subroutines are used. You should group logical graphics operations in subpictures for the sake of modularity and also to help impose an orderly structure on the application program. Although you should "tag" as a subpicture any item that you intend to reference or modify, the use of subpictures is not limited to the expression of graphics elements that are to be repeated frequently in the display file. It is often desirable to use subpictures for reasons other than the obvious benefit of saving space in the display file.

3.1.2 Creating Complete Displays

There are two approaches to creating graphic images. The more conventional approach is to display successive graphic elements on the screen as they are inserted in the display file. This has the effect of creating a "growing" picture on the display screen, one that changes dynamically as new elements are defined.

It may be desirable to use another approach in constructing a picture on the screen. You may want to display a graph or picture only after it has been completely defined. To implement this "all-at-once" technique, define the display as a subpicture that is turned off as soon as it is begun. Then turn the subpicture on again after the definition is complete. This approach is shown below.

```
CALL SUBP (100)
CALL OFF (100)
      .
      .
      .
definition
      .
      .
      .
CALL ESUB(100)
CALL ON (100)
```

Use of this technique also has effect of speeding up the process of image creation.

3.1.3 Attaching a Subpicture

It is possible to move a subpicture around the display screen by attaching it to the tracking object (see the ATTACH subroutine, Section 2.8.4). When you are drawing a subpicture that is to be moved in this fashion, specify a call to APNT (see Section 2.3.1) as the first primitive in the subpicture, but then specify only relative graphic elements as the remaining primitives.

If several instances of the same subpicture are to be moved separately on the screen, then the APNT call should be moved out of the subpicture. You must then issue a separate APNT call for each instance of the subpicture. The APNT call should precede each reference to the subpicture.

3.1.4 Using NMBR for Odometer Output

You may use the NMBR subroutine to create "odometer"-type output (see Section 2.4.7 for a description of NMBR). However, this application normally requires the use of run-time formatting code, which is very costly in terms of memory usage. If you perform simple integer and floating-point output conversions in FORTRAN to accomplish the same task, the use of memory will be much more efficient.

To accomplish the desired task, first convert the number to be displayed to an ASCII string and append a null byte to the string. Then, in order to display the number, issue a CHANGT call (see Section 2.6.6) in a character primitive that you have set up at the beginning of your program.

The brightness of the odometer output can be intensified by means of the INTENS subroutine (see Section 2.7.2) or set to blink mode by means of FLASH (see Section 2.7.3). If the numeric output is in its own subpicture, the size of the characters may also be scaled by issuing a CVSCAL call (see Section 2.4.8).

3.2 GENERAL GRAPHICS TECHNIQUES

The programming techniques described in this section incorporate a variety of miscellaneous approaches to speeding up operations in the DECgraphic-11 system.

3.2.1 Specifying Vector Formats

Vectors may be drawn in any of three legal DECgraphic-11 formats:

- short relative format
- long relative format
- long absolute format

Short vectors require only half the storage of long vectors, but have a limited range. A short vector is stored in one word and a long vector in two words. The range of a short vector is 63 raster units along the x and y axes or approximately one-sixteenth of a full screen. The range of a long vector, on the other hand, is 1023 raster units.

In general, you should use the short vector format when constructing static displays that consist of a large number of short lines (e.g., shading of small objects, very irregular outlines). Short vectors may be altered by means of calls to the GET and CHANGE subroutines (see Sections 2.6.3 and 2.6.4), but not by attaching to the tracking object (see the ATTACH subroutine, Section 2.8.4). Long vector format should normally be used when you are attaching the vector to the tracking object--for example, for the rubber-banding of lines on the display screen.

Absolute vectors should never be used in subpictures that are to be moved as an entity; you should always specify relative vectors in subpicture definitions. Absolute vectors do offer one significantly different feature. Altering one absolute vector does not have the effect of changing the end-point location of the next vector.

When you are uncertain about the precise length of a vector in physical screen units, but are not concerned about subsequently changing the size of the vector, you should use the VECT subroutine (see Section 2.3.3). This routine automatically uses the short vector format whenever possible. Unless short and long vectors alternate frequently, this optimizes storage utilization.

3.2.2 Ordering Display Elements

A display often consists of a number of static elements, such as menus and background, as well as a dynamic picture that grows by repeatedly adding lines, points, and characters to the existing display. When constructing a display of this kind, creating the static portions first is more efficient than using the INSERT subroutine (see Section 2.6.7) to add elements within the display file.

Static elements such as menus may be conveniently defined at the beginning of a subpicture that can be turned off (see Section 3.1.2) and then turned on when needed.

3.2.3 Controlling Display File Size

Using DPTR is often helpful in determining how much space remains in the display file. You may want your program to take appropriate automatic actions when the file reaches a certain size. For example, you might issue an automatic call to CMPRS (see Section 2.9.1) or prevent further expansion when the file is nearly full.

CHAPTER 4

THE RT-11 OPERATING ENVIRONMENT

This chapter describes the operation of the DECgraphic-11 system in an RT-11 environment. It summarizes the contents of the kits supplied in the graphics system, describes procedures for building a library of DECgraphic-11 subroutines, and discusses how to link your programs for use in the graphics system.

4.1 BUILDING THE DECGRAPHIC-11 LIBRARIES

The FORTRAN subroutines used to perform graphics functions in the DECgraphic-11 system are supplied as two kits: a binary kit and a source kit. With these kits, you can build libraries of object modules and source programs by following the procedures outlined below.

4.1.1 Binary Kit

There are three files in the DECgraphic-11 binary kit:

- VT.OBJ for VT11-based systems
- VS.OBJ for single-scope VS60-based systems
- VS2.OBJ for two-scope VS60-based systems

Each is supplied as a file consisting of concatenated object modules of graphics subroutines.

You can build a library of object modules from the VT.OBJ, VS.OBJ, or VS2.OBJ files by using the librarian as shown below (you type the underlined part):

```
.R LIBR <CR>
*GLIB=VT <CR> (or VS or VS2)
```

The librarian will print out several messages of the form:

```
DFILE ILL INS
VTDAT ILL INS
.
.
.
```

You may ignore these messages.

4.1.2 Source Kit

The source kit consists of three files:

- GRPACK.CND
- GRSUBS.MAC
- COND.SAV

GRPACK.CND and GRSUBS.MAC are conditionalized source files from which you can build the graphics library. COND.SAV is a special conditionalizing program used in building the library.

You can build a "customized" library by following the procedure outlined below. You type the text that is underlined in the sample procedure; the system types the rest. An explanation of the questions asked in the sample procedure is included after the output below. The dialog is essentially the same for all versions of DECgraphic-11.

```
.ASSIGN TT:LST <CR>

.ASSIGN TT:LOG <CR>

.LOAD BA <CR>

.R COND <CR>

FILE NAME?
GRPACK <CR>

DO YOU HAVE A VS60 (Y OR N) ?
Y <CR>

TWO SCOPES (Y OR N) ?
N <CR>

ERROR MESSAGE TEXT (Y OR N) ?
Y <CR>

OVERLAID VERSION (Y OR N) ?
N <CR>

STOP --

.R BATCH <CR>
*GRBILD <CR>


$JOB/RT11

*ERRORS DETECTED: 0
FREE CORE: 10556.  WORDS
*
**
DFILE   ILL INS
VTDAT   ILL INS (Repeated many times)
^O        (Type CTRL/O to suppress the display of messages)

$EOJ

END BATCH
```

The various questions asked by the conditionalizer program COND allow you to select specific options if you are building from the source kit. The questions asked during the interaction are described below.

1. DO YOU HAVE A VS60 (Y OR N) ?
   The package may be configured for either a VS60 or VT11 display processor.

2. TWO SCOPES (Y OR N) ?
   If you are configuring for a VS60, it is possible to have a dual-scope system. This question is not asked if the VT11 library is being built and you have answered N to question 1.

3. ERROR MESSAGE TEXT (Y OR N) ?
   You can choose to eliminate the text of the error messages produced by the package in order to save space. In this case, any errors that occur will result in printing an error number that can then subsequently be interpreted by referencing the error message table (see Appendix B).

4. OVERLAID VERSION (Y OR N) ?
   You can produce an overlaid version of your program in order to conserve memory space. Answering Y to this question causes two batch files to be produced, as opposed to one file. The first file (GRBILD) will cause the creation of a set of object files, but will not result in the creation of a library. The second batch file (GRLINK) contains the commands necessary to link your own program to graphics subroutines. The subroutines will be overlaid as much as possible.

Section 4.3 contains complete examples of building libraries for use with the VT11 and VS60.

4.2 LINKING USER PROGRAMS

After you have built a graphics library, you can link your program by issuing the following commands:

```
.R LINK <CR>
*program=program,GLIB/F <CR>
```

where program is the name of your program.

If you have answered Y to question 4 in the COND program, you can link your program in the manner shown below. You type the text that is underlined in the sample procedure.

```
.ASSIGN TT:LOG

.ASSIGN TT:LST

.LOAD BA

.R BATCH
*GRLINK

$JOB/RT11
$EOJ
END BATCH
```

GRLINK is a batch file constructed by COND which will link your program to the DECgraphic-11 subroutines. The contents of GRLINK are of the form:

```
R LINK
USER, USER=USER, subl, sub2, . . . subN
.
.
.
EOJ
```

where subl, sub2, . . . subN is a list of the object files produced by GRBILD, i.e., the DECgraphic subroutines in binary form.

Note that the name USER is automatically given to the loadable program (USER.SAV), the load map (USER.MAP), and the object file corresponding to your program (USER.OBJ). Therefore, you should always name your program's object file USER when you use the compiler:

```
.R FORTRA <CR>
*USER [,USER]= program <CR>
```

where program is the name of your program source file.

After running the batch file GRLINK, you can rename USER.SAV to the name of your choice by using the Peripheral Interchange Program (PIP):

```
.R PIP
*YOUR.NAM = USER.SAV <CR>
```

NOTE

Each time you run GRLINK, old versions of USER.SAV and USER.MAP will be erased.

4.3 PERFORMING USR OPERATIONS

There are a variety of operations that require the RT-11 user service routine (USR) to be swapped into memory (e.g., ASSIGN, CLOSE). When you are performing an operation of this kind in a graphics application, first issue a call to the DECgraphic-11 STOP subroutine to stop the display processor. Call the CONT subroutine to restart the display after the USR operation has completed. (See Sections 2.1.2 and 2.1.3 for a description of these two subroutines.) If you do not follow the procedures outlined above, the display will disappear from the screen and the display processor will hang. The only way to avoid this situation is to issue a SET USR NOSWAP command, a far more core-costly solution. The SAVE and RSTR subroutines require a call to the STOP and CONT subroutines for similar reasons (see Section 2.9.3 for an example).

4.4 SAMPLE PROCEDURES

The following printout illustrates the complete process of building separate libraries for VT11 and VS60 systems.

4.4.1 VT11 Procedures

```
.
.ASSIGN TT:LOG

.ASSIGN TT:LST

.LOAD BA

.R COND

FILE NAME ?
GRPACK

DO YOU HAVE A VS60 (Y OR N) ?
N

ERROR MESSAGE TEXT (Y OR N) ?
Y

OVERLAYED VERSION (Y OR N) ?
N

STOP --

.R BATCH
*GRBILD


$JOB/RT11

*ERRORS DETECTED: 0
FREE CORE: 14648. WORDS

*
**
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
```

```
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
```

```
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
VTDAT    ILL INS


$EOJ

END BATCH
```

4.4.2 VS60 Procedures

```
.ASSIGN TT:LOG

.ASSIGN TT:LST

.LOAD BA

.R COND

FILE NAME ?
GRPACK

DO YOU HAVE A VS60 (Y OR N) ?
Y

TWO SCOPES (Y OR N) ?
Y

ERROR MESSAGE TEXT (Y OR N) ?
N

OVERLAYED VERSION (Y OR N) ?
N

STOP --

.R BATCH
*GRBILD


$JOB/RT11


*ERRORS DETECTED: 0
FREE CORE: 14592. WORDS

*
**
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
```

```
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
DFILE    ILL INS
VTDAT    ILL INS
```

```
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
DFILE   ILL INS
VTDAT   ILL INS
VTDAT   ILL INS

$EOJ

END BATCH
```

CHAPTER 5

THE RSX-11M OPERATING ENVIRONMENT

5.1 INTRODUCATION TO SUPPLIED SOFTWARE

RSX-11M Graphics Extensions, Version 2, provide support for a single VT11 or VS60 display processor operating as a UNIBUS peripheral device. These extensions are a subset of the supplied DECgraphic-11 software and are selected during the question-and-answer dialog generated by the COND program (Section 5.2.1). Answering "no" (N) to the question "RT11?" will result in a subroutine library tailored to the RSX-11M operating system. From the programmer's viewpoint, there is no difference in the operation of DECgraphic-11 between the two operating systems. The subroutine calls are the same, and programs written for one operating system will function identically under the other. Only the initial procedures for building the library and for compiling and linking your program vary. The RSX-11M procedures are discussed in this chapter.

The DECgraphic-11 software kit consists of three files:

COND.FTN, which is a "conditionalizer" program used to tailor the software configuration to your precise needs and hardware;

GRSUBS.MAC, containing the subroutines written in MACRO-11 assembly language; and

GRPACK.CND, which contains those subroutines written in FORTRAN and which creates the indirect command file GRBILD.CMD, used to construct your specialized library.

5.2 OPERATION UNDER RSX-11M

If you are already familiar with the system generation procedure for RSX-11M, you might expect the creation of a DECgraphic-11 library to be complicated under that system. Actually, the procedure is straightforward and will be easy even for someone not familiar with all aspects of RSX-11M. The software kit contains a built-in generating program, COND, which is described fully in Section 5.2.1. Follow the instructions carefully and you will have no difficulty.

Section 5.2 tells you how to turn your FORTRAN program into a "task image" that can be executed by RSX-11M. When you have finished creating the task image, you should study the RSX-11M manuals referenced in the introduction to this book for information on setting task priorities in RSX-11M. The operating system allows you to specify "software priorities" for tasks running in the same environment so that execution time can be allocated properly among them. Depending on the nature of your graphic task, a compromise must

be made between the priority assigned to the graphic task and the priorities given to other tasks running at the same time. For example, testing of a real-time interactive graphic program might reveal that the program is not responding quickly enough to light pen "hits". The "interaction speed" can be increased by resetting the program's software priority to a higher value, but there is a potential problem. If there are other tasks in the environment that require frequent interrupt service, they may be "slowed down" correspondingly as you increase the priority of your graphic routine.

RSX-11M is very flexible and powerful in that you can carefully plan the execution priorities of a virtually unlimited number of tasks; however, the decision to set a task's priority to a certain level requires that you know your system's requirements, not just the requirements of a single graphic program. Fortunately, the procedure for resetting priorities is simple enough to allow for plenty of trials, and as your experience grows you will find the question progressively easier to answer. The referenced RSX-11M Task Builder Reference Manual and Executive Reference Manual are good sources of information about software priorities.

5.2.1 Building Your DECgraphic-11 Library

Although RSX-11M is in general a far more complex (and powerful) system than RT-11, the procedure for building your own customized graphic library is very similar.

When the RSX-11M operating system has been loaded into core, begin the customizing procedure by typing the underlined parts of the following dialog.

```
>FOR COND = COND<CR>
>TKB COND = COND<CR>
>RUN COND<CR>
```

COND.TSK will be executed, responding with a series of questions you must answer yes (Y) or no (N), as shown here:

```
FILE NAME?
GRPACK <CR>
DO YOU HAVE A VS60 (Y OR N)?
Y <CR>              [Answering N implies a VT-11 processor.]
TWO SCOPES (Y OR N)?
N <CR>            [Not asked if answer to above question is N.]
ERROR MESSAGE TEXT (Y OR N)?
Y <CR>
OVERLAID VERSION (Y OR N)?
Y <CR>
RT11 VERSION (Y OR N)?
N <CR>            [Answering N implies an RSX-11M system.]
```

There will be a brief pause while COND is executing, after which it will signal completion by printing

```
TT0--STOP
```

At this point, COND has created a set of new files on your storage device:

GRSUBS.MA, a small file containing your answers to the COND questions, and

GRPAK1.FO, a file of FORTRAN subroutines.

These two files have been assembled by COND to contain only those routines appropriate to your installation, based on your answers to the questions above. Therefore, if you change anything in your system that would result in different answers to the questions, you should run COND again and delete the old versions of GRSUBS.MA and GRPAK1.FO to conserve space on your storage device.

COND also creates an "indirect" command file, GRBILD.CMD, which will generate a library of DECgraphic-11 subroutines that can then be linked to programs you write yourself. To create this library you only have to type the single command shown below:

```
>@GRBILD<CR>
```

You can enter this command in response to the monitor console routine's prompt symbol (> or MCR>) at any time after you have run COND. Usually you will want to enter it right after the "TT0--STOP" message signaling the completion of COND. GRBILD automatically compiles GRSUBS.MA, GRSUBS.MAC, and GRPAK1.FO, putting the resulting object code files together in a new library called GLIB.OLB. GRBILD also does some "housekeeping" tasks that remove old object files, old versions of GRSUBS.MA, and preexisting versions of GLIB.OLB to conserve storage space. When you type @GRBILD, the operating system will print the commands in GRBILD.CMD on the terminal and execute them automatically. You do not need to supply any new information until the message

```
>@EOF
```

appears, signaling the end of the GRBILD indirect command file.

5.2.2 Writing and Running Your Own DECgraphic-11 Programs

PDP-11 FORTRAN is actually a "superset" of ordinary FORTRAN IV in the sense that it allows you more flexibility than the standard language. RSX-11M further relaxes restrictions on FORTRAN programming to some degree. For details of PDP-11 FORTRAN, you should read the PDP-11 FORTRAN Language Reference Manual (DEC order number DEC-11-LFLRA-C-D). Another manual, the IAS/RSX-11 FORTRAN IV User's Guide (DEC-11-LMFUA-C-D) contains helpful information for FORTRAN programmers using RSX-11. In particular, the manual suggests methods for debugging your programs easily and for "optimizing" them (after debugging) so that they will use a minimum of space and execution time. If you are writing programs that will allow an operator to interact with a graphic display in real time, or if your graphic program will be part of an extensive multitask environment, optimization of your program is an especially good practice.

If you are familiar with ANSI FORTRAN, you will find the DECgraphic-11 package simple to use; in effect, the subroutines can be called from your program the same way you would call a subroutine you had written yourself. After writing your program, you must enter it in a source file so that RSX-11M can compile and link it. There are many ways of creating a source file; a common one is to use RSX-11M's EDI (text editing) utility and simply type the FORTRAN statements in from your terminal. For a description of EDI, read the RSX-11M Utilities Procedures Manual (DEC-11-OMUPA-B-D).

After creating your source file, compile the program by typing the underlined parts of the following dialog, inserting the name of your newly created source file in place of SOURCE.FTN, and appropriate names for the output files PROGRAM.OBJ and LISTNG.LST.

```
>FOR PROGRAM.OBJ,LISTNG.LST=SOURCE.FTN <CR>
```

The names for the object and listing files in the command string are "positional", so that if ",LISTNG.LST" is omitted, the FOR compiler will produce only an object file. Similarly, if only ",LISTNG.LST" appears, a listing file will be produced, but no object file.

One feature of PDP-11 FORTRAN that is helpful in debugging your program is the "conditional compile switch", /DE. This compiler feature allows you to insert statements in your program source file with the letter D in column 1 and to then decide at compilation time whether the statements are to be compiled or to be treated as nonexecutable comments. For instance, you could insert such a statement to pass a test parameter to a DECgraphic-11 routine. When you command RSX-11M to compile your program, add "/DE" to the source file, e.g.,

```
>FOR PROGRAM.OBJ,LISTNG.LST=SOURCE.FTN/DE <CR>
```

and the compiler will treat your "D" statement(s) as executable instructions. You can run the program and observe the test results. When you are satisfied that the program is doing its job properly, you can compile the source a second time, this time leaving the /DE switch off. The compiler will generate a new object file in which the D statements are treated as nonexecutable comments.

The last step in this preparation is to take the object file produced by the compiler and build it into an executable task image consisting of the object file linked to the DECgraphic-11 library, GLIB.OLB. Proceed as follows, typing the underlined parts:

```
>TKB <CR>
TKB>TASK.TSK,TASK.MAP=PROGRAM.OBJ <CR>
TKB>GLIB.OLB/LB <CR>
TKB>/ <CR>
ENTER OPTIONS:
TKB>ASG=GR0:1 <CR>
TKB>// <CR>
```

The task builder, TKB, will put the task image in a file called TASK.TSK (or whatever name you put in that position) and a memory allocation map in TASK.MAP. Both of these output files are positional in the same manner as the compiler output files. The options line, ASG=GR0:1, identifies your graphic desplay unit to the operating system. The double slash (//) tells the task builder to begin the building process, after all the files that are to be linked up have been listed (in the example shown, PROGRAM.OBJ and GLIB.OLB). The operating system will respond with the MCR prompt (>) to signal that the task building is finished. Your program is now an executable RSX-11M task that can be run any time by typing

```
>RUN TASK.TSK <CR>
```

The task builder is the "workhorse" of RSX-11M, since it produces the task images toward which the multitask operating system is oriented. As such, it gives the programmer a number of options that can be specified when the compiler output file is made into a task image. Among these options is the ability to assign execution priority to a task. If you are not already familiar with the features of TKB, read the Task Builder Reference Manual (DEC-11-OMTBA-B-D).

If you have generated an overlaid DECgraphic-11 library, you can use the indirect comand file GRBLD.CMD instead of the usual string of TKB command lines. Some RSX-11m systems have been generated in such a way that you are not allowed to enter TKB options on separate lines, in which case using GRBLD.CMD is a necessity. If your system is one of these, you can use GRBLD.CMD to control the task buidler, since the single command line contains all the necessary information for TKB:

```
>TKB @GRBLD <CR>
```

GRBLD.CMD is constructed by COND when you answer Y to the question "OVERLAID?", and it contains two command lines:

```
USER=GRBLD/MP
ASG=GRO:1
```

The second line is the familiar statement which assigns display channel GRO: to logical unit 1. When TKB encounters the first command line and the name GRBLD/MP, it searches for another file containing the overlay description. COND has also created this file and named it GRBLD.ODL. Notice that GRBLD.CMD assigns the task image to a file called USER.TSK; GRBLD.ODL also employs the name USER.OBJ for the root of the overlay. Therefore, if you want to use GRBLD.CMD, you should compile your program source as usual, but name the output file USER. When GRBLD is subsequently run, it will select the most recent version of USER.OBJ, and when the task builder is finished, you can rename the file USER.TSK to a more mnemonic name by using the Peripheral Interchange Program (PIP):

```
>PIP YOUR.NAM[;ver]=USER.TSK <CR>
```

APPENDIX A

DECGRAPHIC-11 SUBROUTINE SUMMARY

This appendix provides an alphabetical summary of the FORTRAN graphics subroutines available in the DECgraphic-11 system. An asterisk (*) before the subroutine name indicates that differences exist between the subroutine calls on the VT11- and VS60-based systems. Usually, the difference is simply that only one scope is supported on the VT11, so the s parameter is ignored in VT11 calls. A cross (+) identifies routines available only in VS60 graphics systems. Calls to these routines are no-ops in VT11 systems. An explanation of the l, i, f, and t parameters referenced in many of these calls is included at the end of this appendix.

Call	Argument List	Effect
ADVANC	(k[,n])	Advances pointer k (1-20) by n primitives from its current position; if n is omitted, advances by one to the next primitive in the display file (see Section 2.6.2).
AGET	(m,j,Z)	Returns in variable Z the jth primitive of the graph or figure subpicture whose tag is m (see Section 2.5.4).
APNT	(x,y[,l,i,f,t])	Positions the beam at the absolute position represented by (x,y) and may display a dot at that position; optionally changes l, i, f, and t parameters (see Section 2.3.1).
APUT	(m,j,b)	Assigns value b to the jth primitive of the graph or figure subpicture whose tag is m (see Section 2.5.5).
+ AREA	(n)	Specifies that subsequent graphics calls refer to the main viewing area (n=1) or the menu area (n=2) (see Section 2.2.2).
* ATTACH	(k[,n,s])	Attaches the primitive referenced by k (1-20) to the tracking object on scope s. When the primitive is a long vector, it will be attached to the object and will follow it; if n is positive or omitted, the vector's origin is stationary and the destination end will move; if n is negative, the destination endi is

Call	Argument List	Effect
		stationary and the origin end will move. If the primitive is an absolute point or vector, n need not be specified (see Section 2.8.4).
+ AVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the absolute point represented by (x,y); optionally changes l, i, f, and t parameters (see Section 2.3.4).
CHANGA	(k,x,y)	Changes the primitive referenced by pointer k (1-20) to the new value specified in (x,y). Like CHANGE, this routine can be used for primitives defined by the routines AVECT, VECT, SVECT, LVECT, RPNT, APNT, XGRA, and YGRA, but for VECT, SVECT, LVECT, and RPNT, subsequent graphics elements are adjusted to be displayed at the same absolute screen position (see Section 2.6.5).
CHANGE	(k,x,y)	Changes the primitive referenced by pointer k (1-20) to the new values specified in (x,y). This routine can be used for primitives defined by the routines AVECT, VECT, SVECT, LVECT, RPNT, APNT, XGRA, and YGRA (see Section 2.6.4).
CHANGT	(k,a1[,a2...])	Changes the character string primitive referenced by pointer k (1-20) to the new string supplied in the call (a1, etc). Characters specified in the call may include control codes indicating italic mode, character rotation, and superscript or subscript mode (see Section 2.6.6).
CMPRS		Compresses the display file by removing all erased primitives and subpictures and reclaiming the space used by them (see Section 2.9.1).
CONT		Restarts the display processor, thus restoring the display interrupted by a call to STOP (see Section 2.1.3).
COPY	([m1],m2)	Creates a copy of the existing subpicture whose tag is m2 and assigns it the tag m1; if the m1 parameter is omitted, subpicture m2 is copied to the currently open subpicture (see Section 2.4.3).

DECGRAPHIC-11 SUBROUTINE SUMMARY

	Call	Argument List	Effect
+	CVSCAL	(m[,ifc,ifv])	Scales the size of displayed characters and vectors in subpicture m; ifc may be in range 1-4, where 1 is one-half normal character size, and 4 is twice normal size (normal is 2); ifv must be in range 1-15, where 1 is one-fourth normal size and 15 is three and three-fourths normal size (normal is 4) (see Section 2.4.8).
*	DETACH	[(s)]	Detaches all primitives from the tracking object on scope s (see Section 2.8.5).
	DPTR	(I)	Returns in I the position of the display file array element in which the next word entered in the display file will be stored (see Section 2.10.1).
	DPYNOP	(n)	Inserts n display no-op instructions at the current position in the display file (see Section 2.10.2).
	DPYWD	(i,j)	Inserts the 16-bit data word (i) in the display file at the current position in the display file; displays the word on the screen if the value of j is zero (see Section 2.10.3).
	ERAS	[(m)]	Erases the definition of subpicture m from the display file; if parameter m is omitted, the tracking object is removed from the screen (see Section 2.4.6).
	ERASP	(k)	Erases the primitive referenced by pointer k (1-20), and repositions the pointer at the next primitive in the display file (see Section 2.6.8).
*	ESUB	[(m)]	Terminates the definition of the current subpicture, created by means of the previous call to SUBP; if the m parameter is included in the call (VS60 only), the display processor status parameters (e.g., light pen enable, intensity) that were in effect before the current subpicture was invoked are restored (see Section 2.4.2).
	FIGR	(A,n,m[,l,i,f,t])	Creates a special figure subpicture m. The figure is plotted from the first n (x,y) coordinate increment pairs specified in array A. Optionally changes l, i, f, and t parameters (see Section 2.5.3).
	FLASH	(k[,f]])	Depending on the value of f, enables or disables flash or blink mode for the primitive referenced by k (1-20) (see Section 2.7.3).

	Call	Argument List	Effect
	FPUT	(m,j,b)	Assigns value b to the jth primitive of the figure subpicture whose tag is m; adjusts the next element of the subpicture to ensure that subsequent points in the figure will be at the same absolute coordinate positions on the screen (see Section 2.5.6).
	FREE		Disconnects the display file from the display processor, thus freeing the area of memory used by the file; this terminates graphics processing until the next call to INIT (see Section 2.1.4).
	GET	(k,X,Y)	Returns in (X,Y) the coordinate positions of the primitive referenced by pointer k (1-20) (see Section 2.6.3).
*	GRID	(gx,gy[,s])	Moves the tracking object on scope s to the nearest point on the grid and automatically detaches any detached primitives. Parameters gx and gy define the spacing of points on the grid (see Section 2.8.6).
	INIT	[(n)]	Clears the display screen, initializes the display file to use the first n words of the FORTRAN NAMED COMMON, DFILE, and sets the initial display file status parameters (see Section 2.1.1).
	INSERT	[(k)]	Reopens the display file for insertion of primitives (specified in subsequent graphics calls). Insertion begins before the primitive referenced by pointer k (1-20). If k is omitted, the insert operation is terminated (see Section 2.6.7).
	INTENS	(k[,i])	Changes the intensity level of the primitive referenced by k (1-20) to the brightness specified in i (1-8) and/or intensifies the referenced primitive (see Section 2.7.2).
	LINTYP	(k[,t])	If t is a legal line type value (1-4), changes the line type of the primitive referenced by k (1-20) to the type specified in t (see Section 2.7.4).
	LPEN	(IH,IT[,X,Y,IP,IA,IM,IT1,IT2])	Indicates whether or not a light pen hit has taken place, and returns information about the hit in the following variables (see Section 2.8.1):

Call	Argument List		Effect
		IH:	nonzero if light pen hit has occurred (1 for scope 1, 2 for scope 2); always 0 or 1 for the VT11.
		IT:	tag of the subpicture in which the hit occurred.
		X,Y:	coordinates of the hit.
		IP:	number of the primitive within the subpicture at which the hit occurred; undefined if the primitive is not in a subpicture.
		IA:	array in which the precedents or ancestors of subpicture IT are stored (subpicture tags in order, starting with innermost nested subpicture).
		IM:	screen area of the light pen hit (1 for main area, 2 for menu area); always 1 for the VT11.
		IT1,IT2:	status of the light pen tip switches for scopes 1 (IT1) and 2 (IT2) (0 for off, 1 for on); for the VT11, IT1 is always 1 and IT2 is always 0.
LVECT	(x,y[,l,i,f,t])		Draws a vector from the current beam position to the relative position represented by (x,y), using the long vector format; optionally changes l, i, f, and t parameters (see Section 2.3.6).
MENU	([x0],y0,dy,m[,'nl'[,'n2']...])		Displays a list of up to 10 items to be used as a menu. The items are the character strings nl, n2, and so on, and the first item is displayed at the coordinate position represented by (x0,y0); if x0 is omitted from a VS60 call, the items are displayed in the hardware menu area of the screen; on the VT11, they begin at the left edge of the viewing area. The dy parameter represents the vertical spacing between menu items. The m parameter represents the tag assigned to the first subpicture (nl); subsequent subpictures are tagged sequentially,

Call	Argument List	Effect
		so if there are 10 subpictures, the 10th subpicture's tag is m+10-1 or m+9 (see Section 2.3.8).
NMBR	(m,var[,n,format])	Creates a special numeric subpicture whose tag is m. This call formats the numeric contents of a FORTRAN variable, var, with a width of n in the specified format. If the format parameter is omitted, the default format of F16.8 is used (see Section 2.4.7).
NOSC		Restores the default unit-scaled coordinate system (x0=0, y0=0) to (x1=1023, y1=1023) for subsequent graphics calls, and eliminates any user-defined coordinate system (see Section 2.2.4).
OFF	(m)	Temporarily turns off the subpicture whose tag is m (see Section 2.4.4).
ON	(m)	Turns on subpicture m (see Section 2.4.5).
POINTR	(k,m[,j])	Sets pointer k to reference the jth primitive of the subpicture whose tag is m; if j is omitted, the pointer will be set to the first primitive of the subpicture. The value of k must be in range 1-20 (see Section 2.6.1).
RPNT	(x,y[,l,i,f,t])	Moves the beam from its current position to the relative position represented by (x,y) and may display a dot at that position. If the current position is (10,20) the beam is moved to (10+x, 20+y). Optionally changes l, i, f, and t parameters (see Section 2.3.2).
RSTR	('[dev:]file[.ext]')	Restores in the memory area of the current display file the display file stored on the mass-storage file identified in the call (see Section 2.9.3).
SAVE	('[dev:]file[.ext]')	Saves the display file by writing it onto the mass-storage file identified in the call (see Section 2.9.2).
SCAL	(x0,y0,x1,y1[,FX,FY])	Defines a new coordinate system in which (x0,y0) identifies the lower left corner of the user-specified screen, and (x1,y1) identifies the upper right corner of the screen; optionally returns the effective X and Y scaling factors in FX and FY (see Section 2.2.3).

DECGRAPHIC-11 SUBROUTINE SUMMARY

	Call	Argument List	Effect
+	SCOPE	(n)	Specifies that subsequent graphics calls refer to scope 1 (n=1) or 2 (n=2) (see Section 2.2.1).
*	SENSE	(k[,l,s])	Depending on the value of l, enables or disables light pen sensitivity for the primitive referenced by k (1-20) in the display file associated with scope s (1 or 2) (see Section 2.7.1).
	STOP		Halts the display processor, stops the display, and clears the display screen. The display may be restored by calling CONT (see Section 2.1.2).
	SUBP	(m1[,m2])	Begins the definition of a subpicture whose tag is m1; all primitives specified in subsequent graphics calls are considered a part of the subpicture, until the occurrence of a terminating call to ESUB. If the m2 parameter is included in the call, a new subpicture tag, m1, references an existing subpicture whose tag is m2 (see Section 2.4.1).
	SVECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position represented by (x,y), using the short vector format; optionaly changes l, i, f, and t parameters (see Section 2.3.5).
	TEXT	(a1[,a2...])	Displays the character strings supplied in the call, (a1, etc.) beginning at the current beam position. Characters specified in the call may include control codes indicating italic mode, character rotation, and superscript or subscript mode (see Section 2.3.7).
*	TRAK	(x,y[,s])	Positions a tracking object on the screen of scope s at coordinate position (x,y) and centers the object on any light pen hit within the tracking area (see Section 2.8.2).
*	TRAKXY	(X,Y[,s])	Returns the coordinates of the current position of the tracking object in (X,Y) for scope s (see Section 2.8.3).
	VECT	(x,y[,l,i,f,t])	Draws a vector from the current beam position to the relative position repesented by (x,y), using the short vector format when possible; optionally changes l, i, f, and t parameters (see Section 2.3.3).
+	WINDOW	(x,Y)	Defines a window on the drawing area of the display screen by specifying the coordinate positions (x,y) of the lower left corner of the user-defined area (see Section 2.2.5).

Call	Argument List	Effect
XGRA	(dy,A,n,m[,l,i,f,t])	Creates a special graph subpicture m, which consists of a series of points. The x values of the points to be plotted are specified in the first n elements of array A, and the y values are given by integral multiples of dy. Optionally changes l, i, f, and t parameters (see Section 2.5.1).
YGRA	(dx,A,n,m[,l,i,f,t])	Creates a special graph subpicture m, which consists of a series of points. The y values of the points to be plotted are specified in the first n elements of array A, and the x values are given by integral multiples of dx. Optionally changes l, i, f, and t parameters (see Section 2.5.2).

ARGUMENT DESCRIPTIONS

Argument	Meaning
l	Light pen enable Default: disabled l=positive. Light pen interaction is enabled, and a light pen interrupt will occur in subsequent graphics output when the light pen is pointed at an object on the display screen. l=0 or omitted. The value of the parameter does not change from its previous status. l=negative. Light pen interaction is disabled, and light pen interrupts will not occur.
i	Intensity level Default: 4; with normal adjustment this makes the primitive light pen-sensitive. i=1 through 8. The intensify status is changed for current graphics output, and the intensity level (brightness of objects on the screen) is set to the value of i, where 1 is the faintest intensity and 8 is the brightest. i=0, omitted or greater than 8. The intensify status is changed for current graphics output, but the level does not change from its previous status for subsequent calls. i=-8 through -1. Current graphics output (except for characters) is not intensified, but the intensity level of subsequent graphics output is set to the absolute value of i.

Argument	Meaning
	i=less than -8. Current graphics output is not intensified, nor is the intensity level changed for subsequent graphics output.
f	Flash mode Default: off
	f=positive. Current and subsequent graphics output is displayed in flash or blink mode.
	f=0 or omitted. The value of the parameter does not change from its previous status.
	f=negative. Flash mode is disabled for current and subsequent graphics output.
t	Line type Default: solid
	t=1. Vectors in current and subsequent graphics output are displayed as solid lines.
	t=2. Vectors are displayed as long-dashed lines.
	t=3. Vectors are displayed as short-dashed lines.
	t=4. Vectors are displayed as dot-dash lines.
	t=0, omitted, negative, or greater than 4. The value of the parameter does not change from its previous status.

APPENDIX B

DECGRAPHIC-11 ERROR MESSAGES

This appendix summarizes all error messages that are generated by the DECgraphic-11 package. For each message, it provides the message number and a brief explanation of the reason for occurrence and references the particular subroutine in which the error occurs.

It is possible to suppress the display of error message text if you are building from the source kit. When you are building your subroutine library (see Chapters 4 and 5), answer N to the question:

```
ERROR MESSAGE TEXT (Y OR N) ?
```

If you have suppressed text output and an error occurs, only the number of the error message will be displayed, in the format shown below:

```
ERROR #YY  XXXXXXXXXX
```

Where YY is the two-digit error message number (see the following list), and the 10-digit X field represents the parameter (e.g., subpicture) in certain messages.

DECgraphic-11 error messages are returned using standard FORTRAN subroutine trace-back logic. Messages are output in the following form:

```
IN ROUTINE "XXXXXX" LINE YYYY
FROM ROUTINE "XXXXXX" LINE YYYY
     .
     .
     .
```

A message of this kind will be preceded by a false error message (number 61):

```
ILLEGAL MEMORY REFERENCE
```

This message can be ignored.

Because many of the DECgraphic-11 subroutines operate internally by invoking other system subroutines, some of the trace-back information will concern subroutine calls that you have not made. You need only be concerned with references to your own subroutine logic.

Number	Error Message	Routine(s)
1	UNABLE TO LINK TO SCOPE You do not have a scope, or it is being used by another task. If you are using RSX-11M, be sure that the option	INIT

Number	Error Message	Routine(s)
	statement "ASG=GR0:1" was included in the task building commands.	
2	INIT NOT CALLED You have not issued an initiating call to the INIT subroutines.	All routines
3	MORE ESUB'S THAN SUBP'S There are more calls to ESUB than corresponding invocations of the SUBP subroutine.	ESUB
4	XXXXX NOT A GRAPH OR FIGR SUBP The element being addressed by an AGET, APUT, or FPUT is not a graph or figure subpicture.	AGET, APUT, FPUT
5	SUBSCRIPT OUT OF RANGE You have specified an array element in an AGET, APUT, that exceeds the the number of entries in the array.	AGET, APUT, FPUT
6	XXXXX NOT A FIGR SUBP The element being addressed by FPUT is not a figure subpicture.	FPUT
7	XXXXX NOT A NMBR SUBP The element being addressed by NMBR is not a numeric subpicture.	NMBR
8	DISPLAY FILE FULL There is insufficient space in the display file to accommodate additional specifications.	All routines
9	FILE TOO BIG The display file being restored from a mass-storage device is too big to fit in the memory area allocated for display file use.	RSTR
10	XXXXX IS IN USE Tag XXXXX is already assigned to a primitive or a subpicture.	SUBP, COPY, XGRA, YGRA, FIGR
11	MORE THAN 8 NESTED SUBP You have specified more than eight nested calls to SUBP.	SUBP
12	XXXXX IS NOT A SUBPICTURE The tag XXXXX is not defined as a subpicture.	SUBP, COPY
13	XXXXX IS STILL OPEN You have attempted to issue a CALL SUBP (m1,m2) for a subpicture that has not been "closed" by means of an ESUB call.	SUBP, COPY

Number	Error Message	Routine(s)
14	ILLEGAL INCREMENT The dx or dy value specified in the call exceeds 63 raster units.	XGRA, YGRA
15	XXXXX ILLEGAL/UNOPEN POINTER The value of pointer XXXXX is not in range 1 through 21, or a referenced pointer has not been defined with a call to POINTR.	All routines that use pointers
16	ILLEGAL DURING INSERT You have attempted to perform an illegal operation while in insert mode.	CMPRS, SAVE, RSTR, SUBP, FLASH, SENSE, INTENS, CVSCAL, ESUB, LINTYP, INSERT
17	TOO FEW ARGUMENTS You have omited one or more required parameters from a subroutine call.	All routines
18	CAN'T COPY NESTED SUBPS You have attempted to copy a subpicture which contains another subpicture.	COPY

APPENDIX C

DISPLAY FILE STRUCTURE

This appendix describes the internal structure of the DECgraphic-11 display file, as it is constructed in memory. You should consult this appendix before attempting to alter a display file or before you change any of the source modules that make up the DECgraphic-11 system.

1. Overall Structure

   a. Root portion

      i. This is controlled by the Handler, which also sets up initial display file status parameters.

   b. User Space

      i. display data, set up by DECgraphic-11 in the COMMON area called DFILE and ending with a DHALT, 0 (173400,0).

2. Subpictures

   a. A reference to an existing subpicture is formatted in the following way for the two display processors:

VT11:	VS60:
173400 (DHALT)	162000 (DJSR)
Address of rest of file	Address of subpicture
Address of subpicture	161002 (DJMPR)
tag	tag
link	link

   b. A subpicture header is formatted as follows:

VT11:	VS60:
Same as for a call	163004 (DJSRR)
	160000 (DJMPA)
	Address of rest of file
	tag
	link

   c. The end of a subpicture has the following format:

VT11:	VS60:
173400 (DHALT)	165000 for ESUB
0	(return, no restore)
	166000 for ESUB (n)
	(return, restore on return)

d. When subpictures are turned off, their formats are changed to the following:

   i. Subpicture references:

      VT11: 160000 (DJMPA) in first word
      VS60: 161004 (DJMPR) in first word

   ii Subpicture definitions:

      VT11: 160000 (DJMPA) in first word
      VS60: 161000 (DJMPR next word) in first word

e. When subpictures are erased, the first two words of the formats are changed to the following:

   VT11 and VS60: 160000 (DJMPA)
   Address of rest of buffer

   The link is also removed from the tag list

f. The pointer to the first tag is in ILST in COMMON VTDAT; the pointer to the last tag (the one that is 0) is in IEND.

3. Display stops are serviced by the Handler.

4. Light pen hits are serviced by the Handler.

5. Display time-outs are serviced by the Handler.

6. Calls to XGRA, YGRA, and FIGR:

   a. XGRA and YGRA have the following format:

      Five-word subpicture header
      Load graphic increment
      Enter graph mode
      X or Y data (one word per point)
      .
      .
      .
      End of subpicture

   b. FIGR has the following format:

      Five-word subpicture header
      164000 (DNOP)
      LVECT
      Data (two words per point)
      .
      .
      .
      End of subpicture

7. Calls to NMBR:

      Five-word subpicture header
      100001 (TEXT MODE)
      Data (two characters per word)
      .
      .
      .
      End of subpicture

8. File structure as a "saved file":

```
Total of n+3 words:
n+2=number of words to follow (first word)
n words of display data
.
.
.
173400,0
```

9. Display file organization on sequential calls to the same module (vector/point/graph plot module):

The example provided below makes sequential calls to the module LVECT.

A single call to LVECT is normally mapped into three words in the display file.

```
word.1 mode (M)
word.2 x coordinate
word.3 y coordinate
```

But on a sequential call to LVECT as indicated below:

```
.
.
.
CALL LVECT(X1,Y1)
CALL LVECT(X2,Y2)
CALL LVECT(X3,Y3)
.
.
.
```

these calls will be mapped into the display file as shown below

```
.
.
.
M
X1
Y1
X2
Y2
X3
Y3
.
.
.
```

This is because the mode word (l,i,f,t) remains unaltered in the second and third call.

APPENDIX D

FORTRAN PROGRAMMING EXAMPLE

The FORTRAN program included in this appendix illustrates the use of all of the graphics subroutines described in Chapter 2. It performs such functions as the following:

- setting up a menu area
- drawing, moving, combining, scaling, copying, erasing, and modifying objects on the display screen
- saving and restoring display files
- using the light pen

This program is included in the DECgraphic-11 distribution. You may want to compile it at your own installation and use it as an aid in testing and demonstrating the graphics capabilities available in the DECgraphic-11 package.

```
      C
      C     SIMPLE DRAWING PROGRAM USING THE NEW GRAPHICS PACKAGE
      C
0001          COMMON/DFILE/IBUF(3500),GRD,NPRIM(200),NVPRIM(200)
0002          REAL SF(4)
0003          LOGICAL*1 FILE1(11),FILE2(11),USED(5)
0004          DATA SF/.25,.5,2.,4./
0005          DATA USED/46,0,0,37,0/
0006          SIZE=3500.
0007          GRD=50.
0008          IWARN=15
0009          MARGIN=10
0010          NDEF=0
0011          NHID=0
0012          ISAC=0
0013          DO 10 I=1,200
0014          NVPRIM(I)=0
0015    10    NPRIM(I)=0
      C
      C     SET UP MENU AREAS
      C
0016          CALL INIT(3500)
0017          CALL SUBP(1000)
0018          CALL OFF(1000)
0019          N=1000./GRD
0020          DO 20 I=0,N
0021          DO 20 J=0,N
0022    20    CALL APNT(I*GRD,J*GRD,,2)
0023          CALL SUBP(1001)
0024          CALL MENU(0.,750.,-50.,2010,'DRAW','MOVE','COMBINE','SCALE',
     X   'COPY','ERASE','MODIFY','HIDE','SEEK')
0025          CALL MENU(0.,300.,-50.,2019,'SEEK & COPY','ROTATE','SAVE',
     X   'RECALL','EXIT')
0026          CALL ESUB
0027          CALL SUBP(1002)
0028          CALL MENU(800.,750.,-50.,1020,'POSITION','LINE','CLOSE','DONE')
0029          CALL ESUB
0030          CALL MENU(800.,750.,0.,1003,'DONE')
0031          CALL SUBP(1004)
0032          CALL MENU(800.,750.,-50.,1030,'1/4','1/2','2 X','4 X')
0033          CALL ESUB
0034          CALL SUBP(1005)
0035          CALL MENU(800.,750.,-50.,1040,'ERASE LINE','SPLIT LINE',
     X   'MOVE CORNER','SHOW ALL','DONE')
0036          CALL ESUB
0037          CALL SUBP(1006)
0038          CALL MENU(800.,750.,-50.,1050,'90 CW','180','90 CCW')
0039          CALL ESUB
0040          CALL SUBP(2000)
0041          CALL APNT(0.,0.,,-4)
0042          CALL TEXT('      ')
0043          CALL APNT(0.,0.,,-4,-1)
0044          CALL ESUB
      C
      C     MAIN LOOP -- WAIT FOR MENU HIT AND BRANCH TO SERVICE IT
      C
0045    100   DO 110 I=1002,1006
```

```
0046  110   CALL OFF(I)
0047        CALL ON(1001)
0048        CALL ON(1000)
0049        CALL DPTR(I)
0050        CALL POINTR(2,2006,2)
0051        I=(SIZE-I)/SIZE*100.
0052        CALL FLASH(2,IWARN-I)
0053        USED(3)=I-I/10*10+48
0054        USED(2)=I/10+48
0055        CALL CHANGT(2,USED)
0056        CALL MENUH(IT,2010,2023)
0057        CALL OFF(1001)
0058        GOTO (1100,1600,1700,1800,1900,2000,2100,2200,2300,2600,2700,
           x  2400,2500,5000),IT
      C
      C     DRAW A NEW OBJECT
      C
0059  1100  IF(I.LT.MARGIN)GOTO 4000
0061        CALL ON(1002)
0062        CALL MAKOBJ(NOBJ)
0063        CALL SUBP(NOBJ)
0064        CALL APNT(500.,500.,,,-4)
0065        CALL POINTR(2,NOBJ)
0066        CALL TRAK(500.,500.)
0067  1110  CALL ATTACH(2)
0068        CALL MENUH(IT,1020,1023)
0069        CALL GRID(GRD,GRD)
0070        CALL GET(2,X,Y)
0071        IF(ABS(X).LT.GRD.AND.ABS(Y).LT.GRD.AND.NPRIM(NOBJ).NE.0)
           x  GOTO 1110
0073        II=-4
0074        GOTO(1140,1120,1160,1180),IT
0075  1120  II=4
0076        NVPRIM(NOBJ)=NVPRIM(NOBJ)+1
0077  1140  NPRIM(NOBJ)=NPRIM(NOBJ)+1
0078        CALL LVECT(0.,0.,,II)
0079        CALL ADVANC(2)
0080        GOTO 1110
0081  1160  II=4
0082  1180  CALL TRAKXY(X,Y)
0083        CALL ERAS
0084        CALL POINTR(2,NOBJ)
0085        CALL GET(2,X0,Y0)
0086        X=X0-X
0087        Y=Y0-Y
0088        IF(ABS(X).LT.GRD.AND.ABS(Y).LT.GRD)GOTO 1185
0090        CALL LVECT(X,Y,,II)
0091        IF(II.GT.0)NVPRIM(NOBJ)=NVPRIM(NOBJ)+1
0093        NPRIM(NOBJ)=NPRIM(NOBJ)+1
0094  1185  CALL ESUB
0095        IF(NVPRIM(NOBJ).EQ.0)GOTO 1190
0097        NOBF=NOBF+1
0098        GOTO 100
0099  1190  CALL ERAS(NOBJ)
0100        NPRIM(NOBJ)=0
0101        GOTO 100
      C
```

```
      C     MOVE AN OBJECT
      C
0102  1600  IF(NDEF.EQ.0)GOTO 100
0104        CALL ON(1003)
0105        CALL PICKOB(IT,2)
0106        CALL POINTR(2,IT)
0107        CALL GET(2,XX,YY)
0108        CALL ATTACH(2)
0109        CALL TRAK(XX,YY)
0110        CALL MENUH(IT,1003,1003)

0111        CALL GRID(GRD,GRD)
0112        CALL ERAS
0113        GOTO 100
      C
      C     COMBINE TWO OBJECTS
      C
0114  1700  IF(I.LT.MARGIN)GOTO 4000
0116        IF(NDEF.LT.2)GOTO 100
0118        CALL PICKOB(IT,2)
0119  1710  CALL PICKOB(IT2,3)
0120        IF(IT2.EQ.IT)GOTO 1710
0122        CALL MAKOBJ(NOBJ)
0123        CALL SUBP(NOBJ)
0124        CALL COPY(,IT)
0125        CALL GET(2,X1,Y1)
0126        CALL GET(3,X2,Y2)
0127        CALL LVECT(X2-X1,Y2-Y1,,-4)
0128        CALL OFF(IT2)
0129        CALL ERASP(3)
0130        CALL COPY(,IT2)
0131        CALL LVECT(X1-X2,Y1-Y2,,-4)
0132        CALL ESUB
0133        NPRIM(NOBJ)=NPRIM(IT)+NPRIM(IT2)+2
0134        NVPRIM(NOBJ)=NVPRIM(IT)+NVPRIM(IT2)
0135        NDEF=NDEF-1
0136        CALL ERAS(IT)
0137        CALL ERAS(IT2)
0138        NPRIM(IT)=0
0139        NPRIM(IT2)=0
0140        NVPRIM(IT)=0
0141        NVPRIM(IT2)=0
0142        GOTO 100
      C
      C     SCALE AN OBJECT
      C
0143  1800  IF(NDEF.EQ.0)GOTO 100
0145        CALL ON(1004)
0146        CALL MENUH(IT2,1030,1033)
0147        CALL PICKOB(IT,2)
0148        CALL OFF(IT)
0149        XX=0.
0150        YY=0.
0151        DO 1830 I=1,NPRIM(IT)
0152        CALL ADVANC(2)
0153        CALL GET(2,X,Y)
0154        CALL CHANGE(2,X*SF(IT2),Y*SF(IT2))
0155        CALL GET(2,X,Y)
```

```
0156          XX=XX+X
0157    1830  YY=YY+Y
0158    1840  CALL GET(2,X,Y)
0159          CALL CHANGE(2,X=XX,Y=YY)
0160          CALL ON(IT)
0161          GOTO 100
        C
        C     COPY AN OBJECT
        C
0162    1900  IF(I.LT.MARGIN)GOTO 4000
0164          IF(NDEF.EQ.0)GOTO 100
0166          CALL ON(1003)
0167          CALL PICKOB(IT,2)
0168    1910  CALL MAKOBJ(NOBJ)
0169          CALL COPY(NOBJ,IT)
0170          CALL POINTR(2,NOBJ)
0171          CALL GET(2,X,Y)
0172          CALL ATTACH(2)
0173          CALL TRAK(X,Y)
0174          CALL MENUH(IT2,1003,1003)
0175          CALL GRID(GRD,GRD)
0176          CALL ERAS
0177          NDEF=NDEF+1
0178          NPRIM(NOBJ)=NPRIM(IT)
0179          NVPRIM(NOBJ)=NVPRIM(IT)
0180          IF(ISAC.EQ.0)GOTO 100
0182          ISAC=0
0183          GOTO 2210
        C
        C     ERASE AN OBJECT
        C
0184    2000  IF(NDEF.EQ.0)GOTO 100
0186          CALL PICKOB(IT,2)
0187          CALL ERAS(IT)
0188          NDEF=NDEF-1
0189          NVPRIM(IT)=0
0190          NPRIM(IT)=0
0191          GOTO 100
        C
        C     MODIFY AN OBJECT
        C
0192    2100  IF(NDEF.EQ.0)GOTO 100
0194          CALL ON(1005)
0195          CALL MENUH(IT2,1040,1044)
0196    2105  IF(IT2.EQ.5)GOTO 100
0198          CALL SENSAL(1)
0199    2110  CALL LPEN(IH,IT,,,IP)
0200          IF(IH.EQ.0.OR.IT.LT.1.OR.IT.GT.200)GOTO 2110
0202          CALL SENSAL(-1)
0203          CALL POINTR(5,IT,IP)
0204          GOTO (2120,2140,2130,2170),IT2
        C
        C     ERASE A LINE
        C
0205    2120  CALL INTENS(5,-10)
0206          NVPRIM(IT)=NVPRIM(IT)-1
0207          IF(NVPRIM(IT).GT.0)GOTO 2100
```

```
0209          CALL ERAS(IT)
0210          NPRIM(IT)=0
0211          NDEF=NDEF-1
0212          GOTO 2100
              C
              C       MOVE A CORNER
              C
0213    2130  IF(IP.NE.NPRIM(IT)+1)GOTO 2150
0215          CALL POINTR(4,IT)
0216          CALL ATTACH(4)
0217          CALL POINTR(6,IT,2)
0218          GOTO 2155
              C
              C       SPLIT A LINE
              C
0219    2140  CALL GET(5,X,Y)
0220          CALL OFF(1000)
0221          CALL CHANGE(5,X/2.,Y/2.)
0222          CALL POINTR(2,IT,IP+1)
0223          CALL INSERT(2)
0224          CALL LVECT(X/2.,Y/2.)
0225          CALL INSERT
0226          CALL ON(1000)
0227          NPRIM(IT)=NPRIM(IT)+1
0228          NVPRIM(IT)=NVPRIM(IT)+1
0229    2150  CALL POINTR(6,IT,IP+1)
0230    2155  CALL ATTACH(6)
0231          CALL ATTACH(6,-1)
0232          CALL POINTR(2,IT)
0233          CALL GET(2,X,Y)
0234          DO 2160 I=1,IP-1
0235          CALL ADVANC(2)
0236          CALL GET(2,XX,YY)
0237          X=X+XX
0238    2160  Y=Y+YY
0239          CALL TRAK(X,Y)
0240          CALL MENUH(IT2,1040,1044)
0241          CALL GRID(GRD,GRD)
0242          CALL ERAS
0243          GOTO 2105
              C
              C       SHOW ALL LINES
              C
0244    2170  CALL POINTR(5,IT)
0245          DO 2180 I=1,NPRIM(IT)
0246          CALL ADVANC(5)
0247    2180  CALL INTENS(5)
0248          NVPRIM(IT)=NPRIM(IT)
0249          GOTO 2100
              C
              C       HIDE AN OBJECT
              C
0250    2200  IF(NDEF.EQ.0)GOTO 100
0252          CALL  PICKOB(IT,2)
0253    2210  CALL OFF(IT)
0254          NVPRIM(IT)=-NVPRIM(IT)
0255          NDEF=NDEF-1
```

```
0256          NHID=NHID+1
0257          GOTO 100
              C
              C       SEEK AN OBJECT
              C
0258    2300  IF(NHID.EQ.0)GOTO 100
0260    2305  DO 2310 I=1,200
0261          IF(NVPRIM(I).LT.0)CALL ON(I)
0263    2310  IF(NVPRIM(I).GT.0)CALL OFF(I)
0265          CALL PICKOB(IT,2)
0266          NVPRIM(IT)=-NVPRIM(IT)
0267          NDEF=NDEF+1
0268          NHID=NHID-1
0269          DO 2320 I=1,200
0270          IF(NVPRIM(I).LT.0)CALL OFF(I)
0272    2320  IF(NVPRIM(I).GT.0)CALL ON(I)
0274          IF(ISAC)1910,100,1910
              C
              C       SAVE THE DISPLAY
              C
0275    2400  CALL INFILE(FILE1,FILE2)

0276          CALL STOP
 GN(1,FILE2)

0278          WRITE(1)NDEF,NHID,NPRIM,NVPRIM
0279          CALL CLOSE(1)
0280          CALL SAVE(FILE1)
0281          CALL LPEN(IH,IT)
0282          GOTO 100
              C
              C       RECALL A DISPLAY FILE
              C
0283    2500  CALL INFILE(FILE1,FILE2)
0284          CALL STOP
0285          CALL ASSIGN(1,FILE2)
0286          READ(1)NDEF,NHID,NPRIM,NVPRIM
0287          CALL CLOSE(1)
0288          CALL INIT
0289          CALL RSTR(FILE1)
0290          CALL LPEN(IH,IT)
0291          GOTO 100
              C
              C       SEEK AND COPY
              C
0292    2600  IF(I.LT.MARGIN)GOTO 4000
0294          IF(NHID.EQ.0)GOTO 100
1003)         CALL ON
0297          ISAC=1
0298          GOTO 2305
              C
              C       ROTATE
              C
0299    2700  IF(NDEF.EQ.0)GOTO 100
0301          CALL ON(1006)
0302          CALL MENUH(IT2,1050,1052)
0303          CALL PICKOB(IT,2)
0304          CALL OFF(IT)
0305          DO 2750 I=1,NPRIM(IT)
```

```
0306          CALL ADVANC(2)
0307          CALL GET(2,X,Y)
0308          GOTO(2710,2720,2730),IT2
0309    2710  CALL CHANGE(2,Y,-X)
0310          GOTO 2750
0311    2720  CALL CHANGE(2,-X,-Y)
0312          GOTO 2750
0313    2730  CALL CHANGE(2,-Y,X)
0314    2750  CONTINUE
0315          CALL ON(IT)
0316          GOTO 100
0317    4000  CALL CMPRS
0318          GOTO 100
0319    5000  STOP
0320          END
```

```
0001          SUBROUTINE SENSAL(I)
      C
      C       TURN LIGHT PEN SENSITIVITY ON/OFF FOR ALL OBJECTS
      C
0002          COMMON/DFILE/IBUF(3500),GRD,NPRIM(200),NVPRIM(200)
0003          DO 100 J=1,200
0004          IF(NVPRIM(J).EQ.0)GOTO 100
0006          CALL POINTR(9,J)
0007          CALL SENSE(9,I)
0008    100   CONTINUE
0009          RETURN
0010          END
```

```
0001          SUBROUTINE MENUM(IT,M1,M2)
      C
      C       WAIT FOR MENU HIT
      C
0002    100   CALL LPEN(IH,IT)
0003          IF(IH.EQ.0.OR.IT.LT.M1.OR.IT.GT.M2)GOTO 100
0005          CALL POINTR(10,IT)
0006          CALL INTENS(10,8)
0007          X=2.
0008          DO 200 I=1,500
0009    200   X=X/X*2.
0010          CALL LPEN(IH,IX)
0011          CALL INTENS(10,4)
0012          IT=IT+1-M1
0013          RETURN
0014          END
```

```
0001            SUBROUTINE PICKOB(IT,IP)
                C
                C       PICK AN OBJECT
                C
0002            COMMON/DFILE/IBUF(3500),GRO,NPRIM(200),NVPRIM(200)
0003            CALL SENSAL(1)
0004    100     CALL LPEN(IH,I)
0005            IF(IH.EQ.0.OR.IT.LT.1.OR.IT.GT.200)GOTO 100
0007            CALL POINTR(IP,IT)
0008            CALL SENSAL(-1)
0009            RETURN
0010            END
```

```
0001            SUBROUTINE INFILE(FILE1,FILE2)
                C
                C       INPUT A FILE NAME
                C
0002            LOGICAL*1 FILE1(11),FILE2(11),DSP(5),DAT(5)
0003            DATA DSP,DAT/'.','D','S','P',0,'.','D','A','T',0/
0004    1       WRITE(5,10)
0005    10      FORMAT(' FILENAME:')
0006            READ(5,20)N,(FILE1(I),I=1,N)
0007            IF(N.EQ.0)GOTO 1
0009    20      FORMAT(Q,6A1)
0010            DO 100 I=1,N
0011    100     FILE2(I)=FILE1(I)
0012            DO 200 I=1,5
0013            FILE1(I+N)=DSP(I)
0014    200     FILE2(I+N)=DAT(I)
0015            RETURN
0016            END
```

```
0001            SUBROUTINE MAKOBJ(NOBJ)
0002            COMMON/DFILE/IBUF(3500),GRO,NPRIM(200),NVPRIM(200)
0003            DO 100 NOBJ=1,200
0004            IF(NVPRIM(NOBJ).EQ.0)RETURN
0006    100     CONTINUE
0007            STOP
0008            END
```

APPENDIX E

DIFFERENCES BETWEEN THE DECGRAPIC-11 AND RT-11 GRAPHICS EXTENSIONS PACKAGES

This appendix summarizes the major differences in subroutine implementation between the DECgraphic-11 FORTRAN support package described in this manual and the RT-11/FORTRAN graphics extensions.

Extensions package subroutines (not supported by DECgraphic-11):

BLNK
SCROL
STAT
TIME
TIMR
UNBLNK

DECgraphic-11 subroutines (not supported by extensions package):

ADVANC	CVSCAL	LINTYP
AREA	DETACH	MENU
ATTACH	ERASP	POINTR
AVECT	FLASH	SCOPE
CHANGA	GET	SENSE
CHANGE	GRID	SVECT
CHANGT	INSERT	TRAKXY
COPY	INTENS	WINDOW

Implementation differences between subroutines supported by both packages:

Subroutine	Differences
AGET, APUT, FPUT, XGRA, YGRA, FIGR	These routines create arrays in Extensions package, subpictures in DECgraphic-11
ESUB	No argument in Extensions package, optional argument in DECgraphic-11
INIT	0-2 arguments in Extensions package, 0-1 arguments in DECgraphic-11
LPEN	More information on light pen hit returned in DECgraphic-11
NMBR	Format width (n) specification allowed in DECgraphic-11
RPNT (RDOT)	RDOT in Extensions package equivalent to RPNT in DECgraphic-11

APPENDIX F

DIFFERENCES BETWEEN DECGRAPHIC-11 AND RSX-11M/FORTRAN GRAPHIC EXTENSIONS

This appendix describes the differences between DECgraphic-11 graphic facilities and the FORTRAN graphic extensions previously available to users of the VT11 display processor and the RSX-11M operating system. The previous extensions have been documented in the RSX-11M/FORTRAN Graphics Extensions User's Guide (DEC-11-AMLEA-A-D), which has been superceded by the present manual.

The following extensions, which were described in the User's Guide, are not provided in the DECgraphic-11 package:

IADRS
STAT
RDOT (function and format same as RPNT in the DECgraphic-11 package)

Similarly, DECgraphic-11 includes the following routines that were not part of the previous set of graphic extensions:

ADVANC	GET
AREA	GRID
ATTACH	INSERT
AVECT	INTENS
CHANGA	LINTYP
CHANGE	POINTR
CHANGT	RPNT (previously called RDOT)
COPY	SCOPE
CVSCAL	SENSE
DETACH	SVECT
ERASP	
FLASH	

In addition to providing the totally new features listed above, DECgraphic-11 also has expanded the scope of certain previously available routines. You should read the appropriate sections of this manual for complete information, but the differences are summarized here for reference:

Subroutine	Differences
ESUB	DECgraphic-11 version allows VS60 users to specify an argument that restores previous display file status parameters.
LPEN	DECgraphic-11 version returns more detailed information on light pen hits.
MENU	DECgraphic-11 version can address the hardware menu area on a VS60 system.
TRAK, TRAKXY	DECgraphic-11 version allows users of a two-scope VS60 to address a specific scope.

GLOSSARY

ABSOLUTE POINT
An individually addressable position on the display screen, identified by specific x- and y-coordinate positions (e.g., x=1023, y=32).

ABSOLUTE VECTOR
A line segment drawn from the current beam position to an absolute point.

ADDRESSABILITY
The smallest discrete unit in which a display element can be defined and to which the hardware responds. The addressability of the total DECgraphic-11 drawing area is one part in 8192 and in the viewing area is one part in 1024.

BEAM
A stream of electrons directed to a position on the display screen; points, vectors, and other graphic elements are generally displayed relative to the current beam position. When you initialize the display processor, the beam is positioned at the lower left corner of the viewing area of the screen (x=0, y=0).

BLINK MODE. See FLASH MODE

BRIGHTNESS. See INTENSITY

CATHODE RAY TUBE (CRT)
An evacuated glass tube in which a beam of electrons is emitted and focused onto a phosphor-coated tube surface. A beam-deflection system moves the beam so that an image is traced out on the surface. The scopes used by the VT11 and VS60 processors are CRT units.

DISPLAY
Contents of the display screen at a given point in time, the result of sequencing through the contents of the display file.

DISPLAY ELEMENT. See PRIMITIVE

DISPLAY FILE
A discrete area of PDP-11 memory that is allocated in the user program and is used to store the graphics instructions and data used in creating displays. The contents of the display file are accessed by the display processor to create images on the screen. It may be saved as an RT-11 file and subsequently restored and used by other programs.

DISPLAY PROCESSING UNIT (DISPLAY PROCESSOR, DPU)
The VT11 or VS60 peripheral graphics unit consisting of a scope and a light pen; the VS60 supports two scopes and a light pen with a tip switch. The DPU accesses instructions and data directly from the display file.

DRAWING AREA.
The total area on which you can define graphic elements. On the VS60, it extends from a lower left corner at coordinate position (x=-4095,y=-4095) to an upper right corner at (x=4095, y=4095) and contains 8192 indivdually addressable positions along the x and y axes. You can define a viewing window on this drawing area in order to examine different portions of it.

FLASH MODE
Mode in which a picture or portion of a picture on the display screen blinks on and off. You may select this mode by specifying a positive integer value for the f parameter (included in many subroutine calls). When you initialize the display file, flash mode is disabled.

FLICKER
An unsteadiness in image intensity caused in refresh displays in which the display processor does not have sufficient time to complete one pass (frame) of the display file, before the phosphor noticeably decays.

FONT
Kind of type in which characters are displayed. You may display characters in the normal type font or in italic mode. To specify italic mode, issue a call to the TEXT subroutine and precede the character string to be displayed with a special control code.

FRAME
One pass of the display processing unit through the display file. To avoid flicker in a refresh display, 30 or more frames per second must typically be executed. After each frame, display file housekeeping is usually performed before returning to the start of the file.

GRAPHIC ARRAY
A collection of values stored in an array and plotted as points on the x- or y-axis by means of the XGRA or YGRA subroutines.

GRID
A logical construct of imaginary points evenly spaced at user-defined intervals on the display screen. If you issue a call to the GRID subroutine and a light pen hit has occurred, the tracking object will be automatically positioned at the point on the screen nearest the light pen hit. This allows you to adjust the coordinates of the hit.

IMAGE-DEFINITION AREA. See DRAWING AREA

INTENSITY
The relative brightness of the graphics output on the display screen. You may select the intensity level of a picture or component of a picture by specifying a new value for the i parameter (included in many subroutine calls) between 1 (faintest) and 8 (brightest). When you initialize the display file, the intensity level is set to 4. If the specified intensity is negative, the display will be invisible. The absolute brightness of a display is dependent on the setting of the intensity control/beam intensity knob on the display processor.

LIGHT BUTTON
A name sometimes given to an entry in the menu area of the display screen. The entry is usually a character string that has been made sensitive to light pen interaction. The entry is normally selected when you touch it with the light pen.

LIGHT PEN
A solid-state, light-detecting device consisting of a photosensitive diode. It is attached by a cord to the VT11 or VS60 display processor. If a primitive or subpicture has been made light pen-sensitive, touching the tip of the light pen to the image on the screen causes a light pen hit to be recorded. Depressing the tip switch on the VS60 light pen provides you with additional interactive control.

LIGHT PEN HIT
An event recorded when the light pen is touched to an image of a primitive or subpicture on the display screen that has been made light pen-sensitive. A hit is internally recognized as an interrupt from the light pen device. Information on the hit is returned by the LPEN subroutine.

LIGHT PEN SENSITIVITY
A characteristic of a primitive or subpicture. If a primitive or subpicture is light pen-sensitive, an interrupt will occur and a hit will be recorded when that image on the display screen is touched with the pen. You may enable light pen sensitivity by specifying a positive value for the l parameter (included in many subroutine calls). When you initialize the display processor, light pen sensitivity is disabled.

LINE TYPE
The type of line used to display vectors on the screen. You may select the type of line from four possible types: 1 (solid), 2 (long-dashed), 3 (short-dashed), 4 (dot-dash). You may specify a new value for the t parmeter (included in many subroutine calls). When you initialize the display file, the line type is solid.

LONG VECTOR
A vector stored in long vector format, occupying two words. A long vector may not exceed 1023 rasters in length.

MAIN AREA
The 12-inch by 12-inch (30-centimeter by 30-centimeter) viewing area of the display screen.

MENU
A list of character string options, sometimes called light buttons, in the menu displayed on the screen. You may select an option from this list by touching the desired character string with the light pen. A maximum of 10 light buttons may be specified in a single invocation of the MENU subroutine.

MENU AREA
A hardware area in the VS60 display processor used to display a list of options called light buttons. The area may be user-specified, but the default is the rightmost 1 1/2-inch vertical strip on the screen (4 centimeters by 30 centimeters). This area can accommodate a horizontal capacity of 128 raster units, usually 14 characters.

POINTER
One of 21 special elements, used to reference primitives in the display file. Pointers are numbered 1 through 21; the 21st pointer is a system pointer and should not be referenced in a user program.

PRECEDENT
In nested subpicture calls, the precedent of subpicture m1 is the "calling" subpicture, i.e., another subpicture which references m1. Because up to eight subpictures can be nested, a subpicture could have as many as seven precedents. In calls to LPEN, the tags of precedent subpictures are returned in array IA.

PRIMITIVE
A basic display element, such as a point, vector, or character string, that can be defined in a single subroutine call and stored in the display file.

RASTER UNIT
The distance between two adjacent addressable points along an axis. There are 1024 x 1024 raster units in the viewing area of the display screen and 8192 x 8192 rasters in the drawing area of the VS60.

RELATIVE POINT
An individually addressable position on the display screen, defined by its relation to the current beam position. If the current beam position is at (10,20), relative point (12,12) is at absolute position (22,32).

RELATIVE VECTOR
A line segment drawn from the current beam position to a coordinate position relative to the beam position.

SCALING
Defining a user coordinate system in which the physical screen coordinate positions are expressed according to a different scale--for example, in increments of ten rather than one. This is accomplished by means of the SCAL subroutine. On the VS60 you may also scale (enlarge or shrink) the size of vectors and characters by means of the CVSCAL subroutine.

SHORT VECTOR
A vector stored in short vector format, occupying one word. A short vector may not exceed 63 rasters in length.

SUBPICTURE
An entity defined by grouping together several primitive definitions. A subpicture is analogous to a subroutine and is used for the same reasons--primarily modularity and efficiency. By referencing a subpicture, you save the overhead required to respecify the primitives included in the subpicture definition.

TAG
A unique name assigned to a subpicture. A tag must be a positive integer in range 1 through 32767.

TIP SWITCH
A switch on the tip of the VS60 light pen that can be depressed when the pen is touched to the screen. It provides additional interactive facilities to the graphics user by allowing an internal switch to be set at the user's option when the light pen is pressed.

TRACKING OBJECT (TRACKING CROSS)
A diamond-shaped image that can be displayed on the screen by means of the TRAK subroutine. It moves automatically to center itself on any light pen hit in its area. If you specify a call to GRID, it will move to the nearest point on the logical grid. The tracking object is stored internally as a subpicture of relative vectors.

UNIT SCALING
The type of scaling in effect in the standard coordinate system, in which adjacent positions on an axis are separated by a single raster unit.

VECTOR
A line segment extending from one coordinate position to another on the display screen. The length of a relative vector may not exceed 1023 raster units.

VIEWING AREA
The current window on the total drawing area of the display processor. On the VS60, the viewing area can be user-defined, but the default is the area extending from a lower left corner at coordinate position (x=0, y=0) to an upper right corner at (x=1023, y=1023). It consists of 1024 individually addressable positions along the x and y axes.

WINDOW
On the VS60, a user-defined segment of the drawing area that is used as the viewing area. A window consists of 1024 x 1024 addressable points. The default window extends from coordinate position (0,0) to (1023,1023).

INDEX

Absolute point, 1-11
Accuracy of light pen hit, 1-16
Addressable points, number of, 2-5
Adjustment of coordinates by GRID, 2-40
ADVANC, 1-24
AGET, 1-23
"all at once" pictures, 2-23, 3-2
APNT, 1-20
APUT, 1-23
AREA, 1-19
Area, drawing, 1-7
Area, menu, 1-8
AREA, repositioning beam for, 2-5
Area of display screen, physical, 1-7
Arguments, omitted subroutine, 1-18
ASG, 5-4
.ASSIGN, 4-2
ATTACH, 1-26
Attaching tracking object to vectors, 2-39
AVECT, 1-20

BATCH, 4-2
Beam, display, 1-11
Beam for AREA, repositioning, 2-5
Beam position, initial, 1-11
Blink, hardware, 1-4
Blink mode, 1-10
Brightness of pictures, 1-3

CHANGA, 1-24
CHANGE, 1-24
Changing character size, 1-13
Changing primitives, 1-13
CHANGT, 1-24
Character, shift-out, 2-14
Character scaling, 1-5
Character scaling factor, 2-25
Character size, changing, 1-13
Character string, 1-11
Characters, extended, 1-4
Characters, "invisible", 2-10, 2-16
Characters, rotated, 1-5
Characters per line, 1-7
Clearing display screen, 2-1
CMPRS, 1-27
CMPRS on pointers, effect of, 2-41
CMPRS on tracking object, effect of, 2-41
CMPRS subroutine, 1-10
Codes, control, 1-11
COMMON block DFILE, 1-9
Compile switch, RSX-11M conditional, 5-4
Compiler, RSX-11M FORTRAN, 5-2, 5-4
Compiler, RT-11 FORTRAN, 4-4
Complex displays, 1-11
COND, 4-2, 5-1, 5-2
COND questions, 4-3
Conditional compile switch, RSX-11M, 5-4
Conditionalizer program, 4-2
Constants, floating-point, 1-3
CONT, 1-19, 4-4
Control codes, 1-11
Conventions, file naming, 2-42
Coordinate reference in subpictures, 2-22
Coordinate system, standard, 1-14
Coordinate system, user-defined, 1-14, 2-7
Coordinates beyond viewing area, 2-6
Coordinates by GRID, adjustment of, 2-40
COPY, 1-22
"customized" library, 4-2, 5-1
CVSCAL, 1-22

Debugging under RSX-11M, 5-3
Definition,
  subpicture, 1-12
DETACH, 1-26
DFILE, 2-2, 2-4
DFILE,
  COMMON block, 1-9
DFILE ILL INS message, 4-1
Direct control of display file, 2-44
Display beam, 1-11
Display elements,
  static, 3-4
Display file,
  direct control of, 2-44
  pointer at end of, 2-31
  saving, 1-11
Display file in use,
  RSTR with, 2-43
Display instructions, 1-10
Display processing unit, 1-9
Display processor, 1-10
Display screen,
  clearing, 2-1
  physical area of, 1-7
Display status parameters, 2-9
Displays,
  complex, 1-11
DPTR, 1-27
DPU, 1-9
DPYNOP, 1-27
DPYWD, 1-27
Drawing area, 1-7
  VS60, 1-14
Duplicated subpicture tags, 2-43

EDI, 5-3
Effect of CMPRS on pointers, 2-41
Effect of CMPRS on tracking object, 2-41
Effect of RSTR on status parameters, 2-44
Effect of SAVE on pointers, 2-43
Effect of SAVE on tracking object, 2-43
End of display file,
  pointer at, 2-31
ERAS, 1-22
ERASP, 1-24
  number of primitive and, 2-34
ESUB, 1-22
  nonrestoring, 2-22
  restoring, 2-21, 2-23
Exit from program, 2-2
Extended characters, 1-4

FIGR, 1-23
Figure,
  "invisible", 2-29
Figure subpicture, 2-28
File format,
  RSX-11M, 2-42, 2-43
  RT-11, 2-42, 2-43
File naming conventions, 2-42
Files,
  program source, 5-3
  using PIP for renaming, 4-4
FLASH, 1-25
Flash mode, 1-10
Floating-point constants, 1-3
FOR, 5-2, 5-4
Format,
  long vector, 2-11, 2-12, 2-13
  short vector, 2-11, 2-12
  vector storage, 3-3
FORTRA, 4-4
FORTRAN compiler,
  RSX-11M, 5-2, 5-4
  RT-11, 4-4
FPUT, 1-23
FREE, 1-19

"garbage collection", 2-41
GET, 1-24
GLIB, 4-1
Graph subpicture, 2-27
GRBILD, 4-3, 5-3
GRBLD, 5-5
Greek letters, 2-14
GRID, 1-27
  adjustment of coordinates by, 2-40
GRLINK, 4-3
"growing" pictures, 2-23, 3-2
GRPACK, 4-2, 5-1
GRSUBS, 5-1

Hardware blink, 1-4
Hardware menu area,
  VS60, 2-18
Hit,
  light pen, 1-17

Image,
  task, 5-1
Image scaling, 1-5
INIT, 1-19
Initial beam position, 1-11
Initial status parameters,
    1-10
INSERT, 1-24
  restrictions on, 2-34
Instructions,
  display, 1-10
INTENS, 1-25
Intensity level, 1-10
  user-selected, 1-3
Intensity levels, 1-3
"invisible" characters,
    2-10, 2-16
"invisible" figure, 2-29
Italic mode, 1-4, 2-16

Letters,
  Greek, 2-14
Levels,
  intensity, 1-3
Librarian,
  RT-11, 4-1
Library,
  "customized", 4-2, 5-1
Light button, 1-16
Light pen hit, 1-17, 2-37
Light pen hit,
  accuracy of, 1-16
Light pen sensitivity, 1-10
Light pen support, 1-6
Light pen tip switch, 1-15
Line,
  characters per, 1-7
Line type, 1-10
Line types,
  user-selected, 1-3
Lines per screen, 1-7
LINK, 4-3
Linker,
  RT-11, 4-3
LINTYP, 1-25
Listings,
  program, 5-4
Logical unit number, 5-5
Long vector format, 2-11,
    2-12, 2-13

LPEN, 1-25
LVECT, 1-21

Mathematical symbols, 2-14
MENU, 1-21
Menu area, 1-6, 1-8
Menu area,
  VS60, 1-9
  VS60 hardware, 2-18
  VT11, 1-9
Mode,
  italic, 1-4
Modularity, 3-1
Moving subpictures, 3-2

Naming conventions,
  file, 2-42
Nested subpictures, 1-12,
    2-20
NMBR, 1-22
  "odometer" output with,
    2-24, 3-2
Nonrestoring ESUB, 2-22
NOSC, 1-19
Number,
  logical unit, 5-5
Number of addressable
    points, 2-5
Number of available
    pointers, 1-13
Number of primitive and
    ERASP, 2-34

"odometer" output with NMBR,
    2-24, 3-2
OFF, 1-22
Omitted parameters, 1-18
Omitted subroutine
    arguments, 1-18
ON, 1-22
Optimizing RSX-11M programs,
    5-3
Options,
  RSX-11M task builder, 5-4
Overlay description,
  RSX-11M, 5-5
Overlays,
  RSX-11M, 5-5

Parameters,
  display status, 2-9
  omitted, 1-18
  status, 1-10

Physical area of display screen, 1-7
Pictures,
"all at once", 2-23, 3-2
brightness of, 1-3
"growing", 2-23, 3-2
PIP, 5-5
PIP for renaming files, using, 4-4
Point,
absolute, 1-11
relative, 1-11
Pointer, 1-13
Pointer at end of display file, 2-31
Pointers, 1-10, 1-12, 2-30
Pointers,
effect of CMPRS on, 2-41
effect of SAVE on, 2-43
manipulating, 1-13
number of available, 1-13
restrictions on, 2-30
POINTR, 1-24
Precedents, 2-38
Primitive and ERASP, number of, 2-34
Primitives, 1-11
Priority,
RSX-11M task, 5-1, 5-4
Procedure,
sample, 4-2
Program,
conditionalizer, 4-2
exit from, 2-2
Program listings, 5-4
Program source files, 5-3
Programs,
optimizing RSX-11M, 5-3

Questions,
COND, 4-3

Raster, 1-7
Raster units,
vector lengths in, 3-3
Redefining window, 1-14
Relative point, 1-11
Removing tracking object, 2-39
Renaming files,
using PIP for, 4-4
Repositioning beam for AREA, 2-5
Restoring ESUB, 2-21, 2-23
Restrictions on INSERT, 2-34
Restrictions on pointers, 2-30
Rotated characters, 1-5
RPNT, 1-20
RSTR, 1-11, 1-27, 4-4
RSTR on status parameters, effect of, 2-44
RSTR with display file in use, 2-43
RSX-11M,
debugging under, 5-3
RSX-11M conditional compile switch, 5-4
RSX-11M file format, 2-42, 2-43
RSX-11M FORTRAN compiler, 5-2, 5-4
RSX-11M overlay description, 5-5
RSX-11M overlays, 5-5
RSX-11M programs,
optimizing, 5-3
RSX-11M task builder, 5-2, 5-4
RSX-11M task builder options, 5-4
RSX-11M task priority, 5-1, 5-4
RT-11 file format, 2-42, 2-43
RT-11 FORTRAN compiler, 4-4
RT-11 librarian, 4-1
RT-11 linker, 4-3

Sample procedure, 4-2
SAVE, 1-11, 1-27, 4-4
SAVE on pointers,
effect of, 2-43
SAVE on tracking object,
effect of, 2-43
Saving display file, 1-11
SCAL, 1-19
SCAL subroutine, 1-14
Scaling, 1-5
character, 1-5
image, 1-5
Scaling factor, 1-19, 2-6
Scaling factor,
character, 2-25
vector, 2-25
SCOPE, 1-19
Screen,
clearing display, 2-1
lines per, 1-7
physical area of display, 1-7
Self-centering of tracking object, 2-39

SENSE, 1-25
Sensitivity,
  light pen, 1-10
Shift-out character, 2-14
Short vector format, 2-11, 2-12
Source files,
  program, 5-3
Special subpictures, 1-11
Special TEXT codes, 2-14
Standard coordinate system, 1-14
Static display elements, 3-4
Status parameters, 1-10
  display, 2-9
  effect of RSTR on, 2-44
  initial, 1-10
STOP, 1-19, 4-4
STOP on VS60, 2-3
Storage format,
  vector, 3-3
String,
  character, 1-11
SUBP, 1-21
Subpicture, 1-11, 1-12
Subpicture,
  figure, 2-28
  graph, 2-27
Subpicture definition, 1-12
Subpicture tag, 1-12
Subpicture tags,
  duplicated, 2-43
Subpictures,
  coordinate reference in, 2-22
  moving, 3-2
  nested, 1-12, 2-20
  special, 1-11
Subpictures vs. subroutines, 3-1
Subroutine arguments,
  omitted, 1-18
Subroutines,
  subpictures vs., 3-1
Subscripts, 1-5, 2-15
Superscripts, 1-5, 2-15
SVECT, 1-21
Symbols,
  mathematical, 2-14

Tag,
  subpicture, 1-12
Tags,
  duplicated subpicture, 2-43
Task builder,
  RSX-11M, 5-2, 5-4
Task builder options,
  RSX-11M, 5-4
Task image, 5-1, 5-4
Task priority,
  RSX-11M, 5-1, 5-4
TEXT, 1-21
TEXT codes,
  special, 2-14
Tip switch, 2-38
  light pen, 1-15
TKB, 5-2, 5-4
Tracking object, 1-16
  effect of CMPRS on, 2-41
  effect of SAVE on, 2-43
  removing, 2-39
  self-centering of, 2-39
  VS60, 2-39
Tracking object to vectors,
  attaching, 2-39
TRAK, 1-26
TRAKXY, 1-26
Type fonts, 1-4

Unit number,
  logical, 5-5
USER, 4-4
User-defined coordinate system, 1-14, 2-7
User-selected intensity level, 1-3
User-selected line types, 1-3
USEREX, 2-2

VECT, 1-20
VECT important use of, 3-3
Vector, 1-11
Vector format,
  long, 2-11, 2-12, 2-13
  short, 2-11, 2-12
Vector lengths in raster units, 3-3
Vector scaling factor, 2-25
Vector storage format, 3-3
Vectors,
  attaching tracking object to, 2-39
Viewing area,
  coordinates beyond, 2-6
VS60,
  STOP on, 2-3
VS60 drawing area, 1-14
VS60 hardware menu area, 2-18
VS60 menu area, 1-9
VS60 tracking object, 2-39

VT11 menu area, 1-9
VTDAT ILL INS message, 4-1

WINDOW, 1-20
Window,
  redefining, 1-14

XGRA, 1-23

YGRA, 1-23

DECgraphic-11
FORTRAN
Reference Manual
DEC-11-GFRMA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Performance Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you require a written reply, please check here. ☐

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754